Nasz nowy adres:
Pl. Inwalidów 10, p. 128 i 136
01-552 Warszawa
tel. 39-14-34 (bez zmiany)

P.1877/90

2 1990

informatyka

Światłowody w systemach informatycznych
Turbo Pascal 5.0

WYDAWNICTWO NOT SIGMA

Cena: 5800 zł

informatyka

nr 2 1990
Luty
Rok wydania XXV.

Miesięcznik
ISSN 0542-9951. INDEX 36124

KOLEGIUM REDAKCYJNE:

mgr Jarosław DEMINET
dr inż. Wacław ISZKOWSKI
mgr Teresa JABŁOŃSKA
(sekretarz redakcji)
Władysław KLEPACZ
(redaktor naczelny)
dr inż. Marek MACHURA
dr inż. Wojciech MOKRZYCKI
dr inż. Wiktor RZECZKOWSKI
mgr Hanna WŁODARSKA
dr inż. Janusz ZALEWSKI
(zastępca redaktora naczelnego)

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab.
Juliusz Lech KULIKOWSKI

Materiałów nie zamówionych
redakcja nie zwraca

WYDAWCA:

Wydawnictwo Czasopism
i Książek Technicznych
NOT SIGMA
Spółka z o.o.
00-950 Warszawa
ul. Biała 4
skr. poczt. 1004

ADRES REDAKCJI:

Pl. Inwalidów 10, p. 128, 136
01-552 Warszawa
tel. 39-14-34

DRUK:

RSW „Prasa-Książka-Ruch"
Prasowe Zakłady Graficzne
ul. Dworcowa 13, 85-009 Bydgoszcz
zam. 191/90
Obj. 4,5 ark. druk. Nakład 6000 egz.

Cena egz. 5800 zł

W sprawach ogłoszeń prosimy zwracać się bezpośrednio do Redakcji.

W numerze:

	Strona
Możliwości stosowania światłowodów w krajowych systemach informatycznych – *Wacław Kubicki, Marian Suskiewicz*	1
Długi wektor w Turbo Pascalu 5.0 – *Henryk J. Runka*	3
Ethernet – od specyfikacji do realizacji praktycznej (1). Specyfikacja – *Mariusz Ormiński*	6
SYS8688 – system pozyskiwania i weryfikacji wiedzy medycznej (1) – *Mieczysław Kłopotek, Adam Kowalski, Maciej Michalewicz, Andrzej Pacan, Elżbieta Syropiatko, Sławomir Wierzchoń, Robert Macura*	9
Propozycja nowego programu uniwersyteckich studiów informatycznych – *Lech Banachowski, Jarosław Deminet, Marek J. Lao*	13
Okres istnienia oprogramowania (2) – *Janusz Zalewski*	18
Język Lotos (2). Specyfikacja procesów – *Zbigniew Huzar, Ludwik Kuźniarz*	21
Dwa kompilatory Moduli 2 dla IBM PC – oprac. *JZ*	26
Ze świata	28
Sieci komputerowe najbardziej dynamiczną gałęzią gospodarki	
European Camac Association	
Terminologia	III s.
Atrybuty jakości oprogramowania – definicje TAEA	okładki

W najbliższych numerach:

- Janusz Zalewski kończy cykl omawiania norm amerykańskich, dotyczących organizacji procesu wytwarzania produktów programowych.
- Lech Banachowski, Jarosław Deminet i Marek J. Lao omawiają szczegółowo propozycję nowego programu uniwersyteckich studiów informatycznych w Polsce.
- Zbigniew Huzar i Ludwik Kuźniarz w trzeciej części charakterystyki języka Lotos opisują sposób specyfikowania systemów.
- Mieczysław Kłopotek i inni przedstawiają sposób działania systemu eksportowego w wybranych obszarach medycyny.
- Mariusz Ormiński zajmuje się naświetleniem praktycznej realizacji sieci Ethernet.

WARUNKI PRENUMERATY

Prenumeratorzy zbiorowi – jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat wyłącznie na blankiecie „wpłata-zamówienie" (jest to „polecenie przelewu" rozszerzone dla potrzeb Wydawnictwa o część dotyczącą zamówienia). Blankiety te będą dostarczane dotychczasowym prenumeratorom przez Zakład Kolportażu. Nowi prenumeratorzy otrzymają je po zgłoszeniu zapotrzebowania (pisemne lub telefoniczne) w Zakładzie Kolportażu, w Radach Wojewódzkich NOT bądź w Redakcjach czasopism.

Prenumeratorzy indywidualni – osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty. Wpłacać należy na konto: **Państwowy Bank Kredytowy III/O Warszawa nr 370015-7490-139-11.**

Prenumerata ulgowa – przysługuje wyłącznie osobom fizycznym – członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły. Sposób zamawiania prenumeraty ulgowej jest taki sam jak prenumeraty indywidualnej. W prenumeracie ulgowej można zamówić tylko po jednym egzemplarzu każdego czasopisma.

Uwaga! Miesięcznik „Aura" może być zamawiany w prenumeracie ulgowej również przez uczniów szkół ogólnokształcących.

Prenumerata ze zleceniem wysyłki za granicę – zamawia się tak, jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy.

Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Wpłaty na prenumeratę przyjmowane są w terminach:
- do 10 listopada na każdy kwartał, I i II półrocze oraz cały rok następny,
- do 28 lutego na II, III i IV kwartał oraz II półrocze,
- do 31 maja na III i IV kwartał oraz II półrocze,
- do 31 sierpnia na IV kwartał.

Zmiany w prenumeracie można zgłaszać pisemnie tylko w wyżej wymienionych terminach.

Informacji o prenumeracie udziela – Zakład Kolportażu Wydawnictwa NOT SIGMA (ul. Bartycka 20, 00-716 Warszawa), skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 wew. 248, 249, 293, 297, 299 lub 40-30-86 i 40-35-89.

Egzemplarze archiwalne czasopism – można nabyć za gotówkę w Klubie Prasy Technicznej, Warszawa ul. Mazowiecka 12 (tel. 26-80-16), lub zamówić pisemnie. Zamówienia na egzemplarze archiwalne czasopism przyjmuje: Zakład Kolportażu, Dział Handlowy, 00-950 Warszawa, skr. poczt. 1004 (tel. 40-37-31), na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena egzemplarza: normalna 5800 zł, ulgowa 1160 zł.

Uwaga! Podane ceny mają charakter wstępny i mogą ulec zmianie, w związku z powyższym Wydawnictwo zastrzega sobie prawo żądania dopłat do już opłaconej prenumeraty.

P.18477/90

WACŁAW KUBICKI,
MARIAN SUSKIEWICZ
Warszawa

Możliwości stosowania światłowodów w krajowych systemach informatycznych

Stosowane obecnie łącza transmisji informacji cechuje dość duży koszt budowy i mała niezawodność. Nie zapewniają one także szeroko rozumianego bezpieczeństwa przesyłanej informacji, tj. jej ochrony przed nieuprawnionym dostępem oraz przed zniekształceniami spowodowanymi oddziaływaniem zakłóceń elektromagnetycznych, generowanych przypadkowo lub niekiedy celowo. Zmusza to projektantów systemów informatycznych do poszukiwania sprzętu tańszego, bardziej niezawodnego, umożliwiającego przesyłanie informacji z dużą szybkością oraz zapewniającego jej ochronę. Szansą są tu łącza budowane przy użyciu światłowodów.

ŁĄCZA ŚWIATŁOWODOWE

Łącza światłowodowe mają wiele właściwości, które uzasadniają celowość ich wykorzystywania do przesyłania informacji w systemach informatycznych. Do zalet łączy światłowodowych można zaliczyć:
- brak promieniowania elektromagnetycznego umożliwiającego nieuprawnione pozyskiwanie informacji przesyłanej przez te łącza;
- brak zakłóceń elektromagnetycznych;
- izolacja elektryczna pomiędzy łączonymi urządzeniami;
- brak przesłuchów między poszczególnymi światłowodami przebiegającymi w tym samym kablu;
- szybkość przesyłania informacji rzędu dziesiątek megabitów na sekundę przy elementowej stopie błędów rzędu $10^{-10} - 10^{-12}$;
- ciężar kabli światłowodowych, duża odporność na szkodliwy wpływ środowiska (odporność na korozję, na działanie kwasów itp.) oraz odporność na działanie zakłóceń elektromagnetycznych generowanych przez pracujące w pobliżu urządzenia elektryczne.

Za stosowaniem łączy światłowodowych przemawiają również względy ekonomiczne. Łącza te są według obecnych cen kilkakrotnie tańsze niż dotychczas stosowane.

Do wad łączy światłowodowych można natomiast zaliczyć:
- mniejszą wytrzymałość mechaniczną (na rozerwanie i na nacisk jednostkowy) kabli światłowodowych niż kabli z rdzeniem metalowym;
- konieczność zachowania większej czystości podczas eksploatacji (wszystkie poiączenia rozłączne są wrażliwe na zabrudzenia, wskutek czego zwiększa się tłumienność łącza);
- konieczność stosowania specjalistycznego sprzętu pomocniczego (spawarki do światłowodów lub zestawy do ich klejenia, mierniki tłumienności itp.).

W celu przesyłania informacji za pośrednictwem światłowodów tworzy się tzw. **trakty światłowodowe**, których podstawowymi elementami są:
- światłowód,
- nadajnik promieniowania,
- odbiornik promieniowania.

Nadajnik oprócz źródła promieniowania zawiera układy elektroniczne dopasowujące sygnały elektryczne napływające z zewnątrz (np. z komputera) do postaci wymaganej przez źródło promieniowania. Jako źródła promieniowania stosuje się diody elektroluminescencyjne LED i lasery półprzewodnikowe emitujące promieniowanie podczerwone o określonej długości fali. W odbiorniku oprócz fotodetektora (dioda PIN) znajdują się układy elektroniczne wzmacniające i formujące sygnał wyjściowy do wymaganego poziomu (np. do poziomu TTL).

Oprócz tych elementów w skład traktu światłowodowego mogą wchodzić elementy dodatkowe, ułatwiające eksploatację łączy lub umożliwiające pełniejsze ich wykorzystanie. Elementami tymi mogą być m.in.:
- złącza rozłączne,
- sprzęgacze optyczne,
- modulatory i demodulatory,
- filtry optyczne.

W celu dołączenia nadajnika i odbiornika promieniowania do źródła i ujścia danych niezbędne są układy pośredniczące dopasowujące parametry nadajnika i odbiornika promieniowania do parametrów źródła i ujścia danych (np. do wymagań sprzęgu RS 232C). Trakt światłowodowy wraz z układami pośredniczącymi nazywa się zwykle **łączem światłowodowym**.

ZASTOSOWANIE ŚWIATŁOWODÓW W SYSTEMACH INFORMATYCZNYCH

Ubogi asortyment elementów optoelektronicznych oferowanych na rynku krajowym (np. brak w handlu rozgałęziaczy i sprzęgaczy optycznych) powoduje, że w chwili obecnej łącza światłowodowe zbudowane z dostępnych na rynku elementów można stosować do przesyłania informacji jedynie w relacji dwupunktowej (punkt–punkt). Takie łącza mogą być stosowane w systemach abonenckich i sieciach komputerowych: lokalnych i rozległych terytorialnie. W systemach abonenckich łącza światłowodowe mogą być wykorzystywane do połączenia terminali z komputerem centralnym. Terminalami są tu zwykle monitory z klawiaturą, plotery lub mikrokomputery, połączone najczęściej przez sprzęg RS 232C. Terminale w systemach abonenckich są zlokalizowane zwykle w odległości od kilkudziesięciu do kilkuset metrów od komputera centralnego (podobne odległości są między elementami lokalnych sieci komputerowych).

Znajdujący się na krajowym rynku sprzęt optoelektroniczny umożliwia budowanie światłowodowych łączy transmisji danych, nadających się do stosowania w lokalnych sieciach komputerowych o dostępie niekolizyjnym. Zastosowanie takich łączy w lokalnych sieciach komputerowych o dostępie rywalizującym wymaga stosowania bardziej rozbudowanych układów pośredniczących, dopasowujących parametry traktu światłowodowego do parametrów źródła i ujścia danych (istnieje potrzeba wydzielenia w łączu światłowodowym kanału do przesyłania sygnału kolizji). Ze względu na stosunkowo niewielkie długości łączy światłowodowych stosowanych w systemach abonenckich i lokalnych sieciach komputerowych, do budowy tych łączy mogą być stosowane tańsze światłowody o skokowo zmiennym współczynniku załamania światła, tzw. światłowody *„step index"*.

Inaczej przedstawia się sprawa zastosowania łączy światłowodowych w rozległych terytorialnie sieciach komputerowych. Ze względu na duże odległości między elementami tych sieci, do budowy łączy transmisji danych należałoby stosować droższe światłowody, tzw. **gradientowe**, o mniejszym współczynniku tłumienia, oraz regeneratory sygnałów optycznych zapewniające odpowiedni poziom odbieranych sygnałów. Należy również pokonać wiele trudności technicznych związanych z ułożeniem tak długich łączy, z zasilaniem regeneratorów itp. Wydaje się, że łącza światłowodowe w tego rodzaju sieciach komputerowych będą mogły być stosowane dopiero po zbudowaniu publicznej sieci łączności na światłowodach. Jest natomiast celowe i bardzo potrzebne już obecnie wykorzystywanie łączy światłowodowych w systemach abonenckich i lokalnych sieciach komputerowych. Wymaga to jednak opracowania i zbudowania odpowiednich układów pośredniczących; jednym z nich jest konwerter światłowodowy KS-1.

KONWERTER ŚWIATŁOWODOWY KS-1

W Wojskowym Instytucie Informatyki zaprojektowano, zbudowano i przebadano konwerter światłowodowy KS-1 umożliwiający dwukierunkową transmisję danych przez sprzęg RS 232C. Przy opracowywaniu koncepcji konwertera przyjęto następujące założenia:
- konwerter powinien umożliwiać wymianę informacji w relacjach terminal–komputer i komputer–komputer przez sprzęg RS 232C;

- zastosowanie konwertera nie powinno wymuszać dokonywania zmian w eksploatowanym standardowym sprzęcie komputerowym;
- konwerter powinien umożliwiać eksploatowanie wykorzystywanego obecnie oprogramowania podstawowego i usługowego;
- urządzenie powinno być kodowe przezroczyste;
- do budowy konwertera należy wykorzystać elementy produkowane w krajach RWPG.

Konwerter KS-1 składa się z następujących zespołów funkcjonalnych:
– układów optoelektronicznych,
– interfejsu,
– zasilania,
– alarmu.

Zespół układów optoelektronicznych składa się z modułu nadajnika optoelektronicznego WK-164-41 i modułu odbiornika optoelektronicznego WK-164-42 (podstawowe dane techniczne tych modułów podano w tabeli). W nadajniku optoelektronicznym następuje zamiana sygnałów elektrycznych (poziom TTL) na sygnały świetlne o długości fali świetlnej 820 nm. Sygnały te są wprowadzane przez złącze rozłączne do światłowodu połączonego z odbiornikiem optoelektronicznym. W odbiorniku WK-164-42 znajduje się monitor kontrolujący stan linii światłowodowej. Wysoki poziom napięcia na końcówce 8. odbiornika oznacza, że linia jest sprawna, niski poziom natomiast oznacza, że linia jest przerwana lub ma dużą tłumienność. Monitor ten zastosowano w konwerterze do sterowania układem alarmu.

Dane techniczne modułów nadajnika WK-164-41 i odbiornika WK-164-42 (szybkość transmisji: 0.40 Mb/s, temperatura pracy: −10°..+50°C, długość fali: 820 nm, zakres działania ARW: 30 dB)

Nadajnik WK-164-41		Odbiornik WK-164-42	
Końcówka	**Sygnał**	**Końcówka**	**Sygnał**
1	Masa	1	Masa
2	Wejście TTL	2	NC
3	NC	3	NC
4	Kod wewnętrzny	4	$V_{pp} = -5V$
5	Masa	5	Masa
6	Masa	6	Masa
7	$V_{cc} = +5$ V	7	$V_{cc} = -5V$
8	NC	8	Wyjście monitorowe
9	NC	9	Wyjście TTL Q
10	Masa	10	Wyjście TTL Q

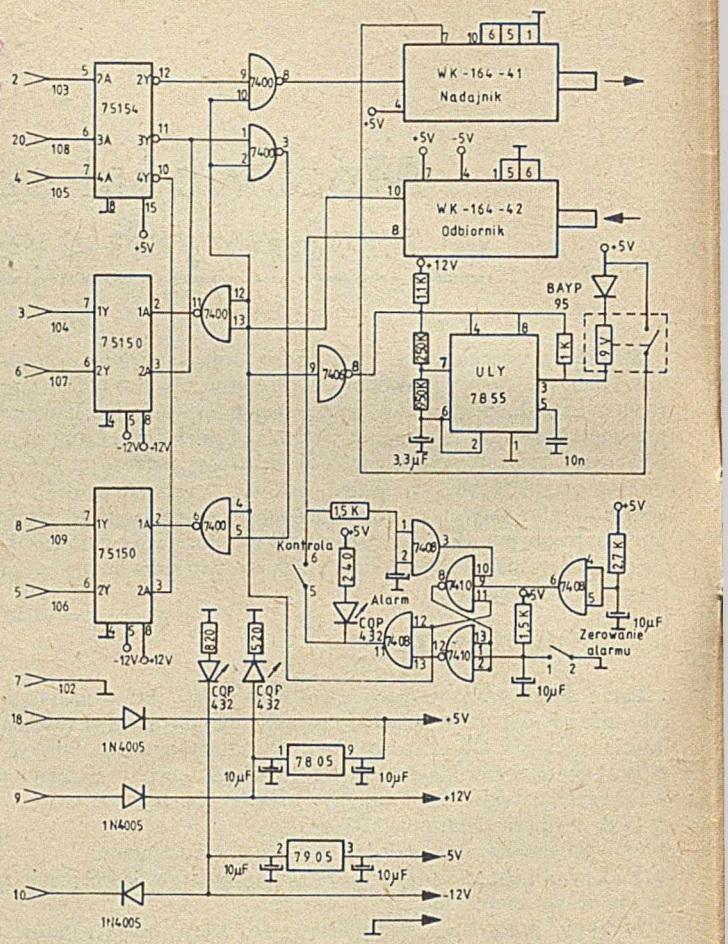
Zespół interfejsu ma zadanie dopasowania sygnałów TTL (0–5 V) do poziomu wymaganego przez sprzęg RS 232C (+ 12 V). Zespół interfejsu jest zbudowany z użyciem układów 75150 i 75154. Przy współpracy konwertera z komputerem (terminalem) mogą być wykorzystywane linie od 101 do 109 sprzęgu RS 232C. Linie te są wyprowadzone na 25-stykowe złącze szufladowe Eltra 871025 lub Eltra 881025.

Zespół zasilania składa się ze stabilizatorów napięcia +5 V i −5 V, diod blokujących i diod LES sygnalizujących doprowadzanie napięć +12 V i −12 V. Konwerter może być zasilany z zasilacza zewnętrznego lub przez sprzęg RS 232C z komputera lub terminala. Zasilanie konwertera przez sprzęg RS 232C wymaga doprowadzenia na styk 9. napięcia +12 V, a na styk 10. – napięcia −12 V. Pobór prądu wynosi 300 mA dla +12 i 100mA dla −12 V.

Zadaniem **zespołu alarmu** jest przerwanie transmisji w przypadku niedrożności trasy optycznej (np. przerwanie światłowodu), zwiększenia tłumienności tej trasy (np. przy próbie założenia podsłuchu) lub zaniku zasilania w którymś ze współpracujących konwerterów. Oprócz przerwania transmisji następuje w tych przypadkach zdjęcie sygnału gotowości konwertera do nadawania, a jednocześnie zapala się dioda sygnalizująca alarm. Sygnał alarmu w jednym konwerterze powoduje uruchomienie bloku alarmu w konwerterze współpracującym (u odbiorcy). Ponowne uruchomienie konwertera powinno być poprzedzone analizą przyczyny alarmu. Wciśnięcie przycisku *„Zerowanie alarmu"* w jednym z konwerterów powoduje ponowne uruchomienie łącza. Zastosowanie w konwerterze KS-1 tak rozbudowanego bloku alarmu wynika z faktu, że konwerter ten jest wykorzystywany do przesyłania informacji wymagającej szczególnej ochrony.

Konwerter KS-1 został wykonany jako moduł wolno stojący o wymiarach 140 × 100 × 40 mm i wadze 400 g.

Schemat konwertera przedstawiono na rysunku. Konwerter KS-1 jest łączony z komputerem lub terminalem za pomocą krótkiego kabla.



Schemat konwertera KS-1

Zapobiega to wyłamywaniu złącza na pakiecie sprzęgu szeregowego. Przedstawiony układ może być znacznie uproszczony. W niektórych systemach można całkowicie zrezygnować z bloku alarmu. Jeżeli do konwertera zostaną doprowadzone napięcia ±12 V i ±5 V, to nie będzie potrzebny również blok zasilania. Niektórym użytkownikom wystarczy, aby w konwerterze były aktywne tylko linie sterujące 102, 103 i 104, i odpowiednio zwarte pozostałe linie sprzęgu RS 232C. Spowoduje to znaczne uproszczenie konwertera, zmniejszenie jego wymiarów i kosztów budowy łącza światłowodowego.

WYNIKI BADAŃ FUNKCJONALNYCH KONWERTERÓW KS-1

W celu przeprowadzenia badań funkcjonalnych konwerterów KS-1 zestawiono stanowisko badawcze składające się z dwóch mikrokomputerów IBM PC i dwóch konwerterów KS-1 połączonych dwoma kablami światłowodowymi o długości po 300 m. W czasie badań sprawdzono:
– możliwość poprawnego przesyłania informacji z szybkością od 59 bitów/s do 115 000 bitów/s;
– wielkość elementowej stopy błędów łącza;
– możliwość korzystania ze standardowych programów transmisji;
– poprawność i skuteczność działania bloku alarmu.

Możliwość przesyłania informacji z różnymi szybkościami i określanie elementowej stopy błędów sprawdzono przy użyciu specjalnie przygotowanego programu. Konwerter umożliwia przesyłanie informacji z każdą z badanych szybkości, a przy przesyłaniu na każdej z wybranych szybkości po ok. 10 000 000 znaków nie stwierdzono błędów transmisji. Sprawdzono również możliwość wymiany informacji między mikrokomputerami przy użyciu standardowych programów do transmisji plików KERMIT i CROSSTALK. Transmisja przebiegała poprawnie. Przy badaniu konwertera wykorzystano również pakiet programowy LAN-Link do tworzenia sieci mikrokomputerowych.

dokończenie na s. 5

HENRYK J. RUNKA
Instytut Cybernetyki Ekonomicznej
Akademia Ekonomiczna
Poznań

Długi wektor w Turbo Pascalu 5.0

W artykule przedstawiono, w jaki sposób można operować w Turbo Pascalu 5.0 wektorem o liczbie składowych ograniczonej tylko przez dostępną wolną pamięć komputera.

W Turbo Pascalu 5.0 zarówno segment danych, jak i blok pamięci przydzielonej jednemu obiektowi (zmiennej) w tym segmencie nie może przekroczyć 64 KB (dokładnie – 65 520 bajtów). Natomiast obszar pamięci przydzielany obiektowi z wolnej pamięci, tzw. sterty (ang. *heap memory*), nie może przekroczyć 65 521 bajtów. Przy tym, aby uniknąć fragmentacji tego obszaru, należy przydzielać pamięć blokami o rozmiarach, które są wielokrotnościami liczby 16, co odpowiada rozmiarowi paragrafu w wersji 3.0. Zatem i w tym przypadku, maksymalny blok pamięci nie powinien przekraczać 65 520 (tj. 4095 × 16) bajtów. Stosując tę zasadę przydziału pamięci, za pomocą procedury

GetMem (var p:pointer; size:word)

gdzie *p* jest zmienną typu wskaźnikowego, a *size* – rozmiarem bloku, zmiennej *p* jest przypisany znormalizowany wskaźnik (adres) postaci *hhhh*:0000, gdzie *h* oznacza cyfrę szesnastkową.

Jeśli możliwy jest przydział sąsiednich bloków pamięci o rozmiarach 65 520 bajtów (FFF0H), z wyjątkiem ostatniego bloku, którego rozmiar może być mniejszy, to adresy przypisane zmiennym wskaźnikowym różnią się dla segmentu o 4095 (0FFFH), natomiast wyrównanie (ang. *offset*) jest ustawione na zero. W ten sposób można potraktować kolejne bloki pamięci jako jeden blok przydzielony tylko jednemu obiektowi. Proponowane przełamanie granicy 64 KB przydziału pamięci dla obiektów może być wykorzystane przy tworzeniu wektorów zawierających składowe (elementy) dowolnego typu. Takiego wektora nie można jednak indeksować w tak łatwy sposób, jak w przypadku tablicy jednowymiarowej, zmiennej strukturalnej typu *array*.

Na wydruku 1 przedstawiono jednostkę (ang. *unit*) LongRArr, zawierającą definicje typów i implementację operacji, za pomocą których można przydzielać pamięć „długiemu wektorowi" oraz operować *i*-tym elementem tego wektora. W jednostce LongRArr przyjmuje się, że elementem wektora jest liczba zmiennoprzecinkowa (typu *real*). Jeśli elementami wektora mają być inne obiekty, to należy przedefiniować typ **elementtype**.

Bardziej elastyczne jest jednak korzystanie z dwóch plików:
- LongArr.typ, zawierającego definicje typów (wydruk 2),
- LongArr.ope, zawierającego implementację operacji na wektorze, która może być podobna do implementacji przedstawionej na wydruku 1, w części występującej po **implementation**.

```
type
     Ptr_to_long     =^longint;
     Ptr_to_int      =^integer;
     Ptr_to_word     =^word;
     Ptr_to_real     =^real;
     Ptr_to_short    =^shortint;
     Ptr_to_byte     = ^byte;
     Ptr_to_char     = ^char;
     Ptr_to_extended=^extended;
     Ptr_to_double   =^double;

     ptr_to_element=^elementtype;
     ith_arr_element=longint;

const
      BlockSize:word=$FFF0;
      SegIncr:word  =$0FFF;

type
     PtrStruc=record
       ofs,seg:word;
     end;
```

W głównej części programu należy wówczas zdefiniować typ **elementtype**, a następnie włączyć zawartość pliku LongArr.typ, przez opcję kompilatora

{$I LongArr.typ}

W obszarze procedur i funkcji programu należy włączyć drugi plik przez opcję

{$I LongArr.ope}

```
program LongArray;

 uses mWindowU,LongRArr,CRT;

procedure Use_Long_Vector;

 var
     i : longint; r : integer;
     base_ptr : pointer;
     vW : UserWindows;
     ch : char;
     VideoMode : byte absolute $0040:$0049;
     n,size : longint;
     maxRow : integer;
     vm : VideoModeType;

 const
       first_window  : integer = 1;
       second_window : integer = 2;

 begin
   { Poniższe funkcje i procedury                                    }
   vm:=SET_VIDEO_MODE(VideoMode);
   INIT_STATUS_OF_WINDOWS( vW,3 );
   SET_TEXT_WINDOW( vW,first_window,'',1,1,last_Screen_Col,5,
                    30,2,SingleBrd,
                    LightMagenta,LightGray,
                    LightMagenta,Green,
                    ClearOn,DisplayOn                                );
   DISPLAY_TITLE( vW,first_window,'Large Vector',Centered);
   SET_TEXT_WINDOW( vW,second_window,'',1,6,
                    last_Screen_Col,last_Screen_Row-1,
                    1,1,DoubleBrd,
                    Yellow,Black,
                    Green,Red,
                    ClearOn,DisplayOn                                );
   { zawiera jednostka mWindowU                                      }

   maxRow:=HI(WindMax)-HI(WindMin);
   repeat
     write('How many real numbers? '); readln(n);
     if n<=0 then Exit;
     base_ptr:=ALLOC_MEMORY(n);
     if base_ptr=NIL then begin
       writeln('Memory not allocated!');
       Halt(1);
     end;
      for i:=1 to n do begin
       write('number ',i:6,' is ' );
       readln(ith_position(base_ptr,i)^);
     end;                                                            }
     for i:=1 to n do
      ith_position(base_ptr,i)^:=i;
     r:=0;
     for i:=1 to n do begin
       write(ith_element(base_ptr,i):8:2,' ' );
       if i mod 7 =0 then begin
       writeln; r:=r+1;
       if r>=maxRow then begin
         Wait_to_Go; writeln;
         r:=0;
       end
     end;
   end;
   if i mod 7 <>0 then writeln;
   FREE_MEMORY(base_ptr,n);
   until false;
end;

var
    vm:VideoModeType;

begin
  Use_Long_Vector;
  Wait_to_Go;
  vm:=SET_VIDEO_MODE(LastMode);
  NormVideo;
end.
```

```
unit LongRArr;

interface

uses CRT;

type
     elementtype=real;
     ptr_to_element=^elementtype;
     ith_arr_element=longint;


{$F+} function HeapFunc( Size:word ):integer; {$F-}
{
    Zmienia standardowe działanie, w przypadku, gdy nie można
    przydzielić bloku pamięci z obszaru heap - wartość NIL jest
    przypisana zmiennej wskaźnikowej.
}

function Ptr_increment( p:pointer; size:word ):pointer;
{
    Dostarcza wskaźnik do następnego elementu wektora.
}

function MyGetMem( size:word ):pointer;
{
    Dostarcza wskaźnik do bloku pamięci.
}

procedure MyFreeMem( p:pointer; size:word );
{
    Zwalnia blok pamięci.
}

function ALLOC_MEMORY( n:longint ):pointer;
{
    Dostarcza wskaźnik do pierwszego bloku z ciągu bloków pamięci,
    bezpośrednio po sobie następujących.
}
procedure FREE_MEMORY( p:pointer; n:longint );
{
    Zwalnia bloki pamięci przydzielone przez ALLOC_MEMORY.
}
function ith_position( base_ptr:pointer; i:ith_arr_element ) :
                                                ptr_to_element;
{
    Dostarcza wskaźnik do i-tego elementu wektora.
}

function ith_element( base_ptr:pointer; i:ith_arr_element ) :
                                                 elementtype;
{
    Dostarcza wartość i-tego elementu wektora.
}

procedure ith__element( base_ptr:pointer; i:ith_arr_element;
                        var v_i:elementtype              );
{
    Wartość i-tego elementu jest przypisana zmiennej v_i . Należy
    używać tę procedurę w przypadku, gdy element wektora jest typu
    strukturalnego.
}

procedure Wait_to_Go;
{
    Zatrzymuje wykonanie programu, dopóki nie zostanie naciśnięty
    dowolny klawisz klawiatury.
}

implementation

 const
       BlockSize: word=$FFF0;
       SegIncr:word  =$0FFF;
       MaxBlocks     =25;

 type
      PtrStruc=record
        ofs,seg:word;
      end;


function HeapFunc( Size:word ):integer;
  begin HeapFunc:=1 end;

function Ptr_increment( p:pointer; size:word ):pointer;

 var
    new_ofs:longint;

 begin
   if longint(PtrStruc(p).ofs+size)>=BlockSize then begin
     new_ofs:=longint(PtrStruc(p).ofs)+longint(size)-BlockSize;
     Inc(PtrStruc(p).seg,SegIncr);
     PtrStruc(p).ofs:=word(new_ofs);
   end
   else Inc(PtrStruc(p).ofs,size);
   Ptr_increment:=p;
 end;

function MyGetMem( size:word ):pointer;
 var
     p:pointer;

 begin
   System.GetMem(p,(size+$F) and $FFF0);
   MyGetMem:=p;
 end;

procedure MyFreeMem( p:pointer; size:word );

 begin
   System.FreeMem(p,(size+$F) and $FFF0);
 end;

function ALLOC_MEMORY( n:longint ):pointer;

 var
     p:pointer;
     vptr:array[1..MaxBlocks] of pointer;
     nBlocks,i:integer;
     contiguous_memory,memalloc:boolean;
     size:longint;
     lastBlockSize:word;

 begin
   size:=(n+1)*Sizeof(elementtype);
   nBlocks:=(size-1) div BlockSize+1;
   lastBlockSize:=word(size-(nBlocks-1)*BlockSize);
   if size>MemAvail then begin
     writeln('Not enough memory!');
     ALLOC_MEMORY:=NIL;
     Exit;
   end;
   if nBlocks>maxBlocks then begin
     writeln('Too many blocks of memory required!');
     ALLOC_MEMORY:=NIL;
     Exit;
   end;
   for i:=1 to nBlocks do vptr[i]:=NIL; memalloc:=true;
   for i:=1 to nBlocks-1 do begin
     vptr[i]:=MyGetMem(BlockSize);
     if vptr[i]=NIL then memalloc:=false;
   end;
   vptr[nBlocks]:=MyGetMem(lastBlockSize);
   if vptr[nBlocks]=NIL then memalloc:=false;
   if not memalloc then begin
     for i:=1 to nBlocks-1 do
       if vptr[i]<>NIL then
         MyFreeMem(vptr[i],BlockSize);
     if vptr[nBlocks]<>NIL then
         MyFreeMem(vptr[nBlocks],lastBlockSize);
     ALLOC_MEMORY:=NIL;
     Exit;
   end;
   contiguous_memory:=true;
   for i:=1 to nBlocks-1 do
     if PtrStruc(vptr[i]).seg+SegIncr<>
        PtrStruc(vptr[i+1]).seg then begin
       contiguous_memory:=false;
       writeln('Not contiguous memory allocated!');
       writeln('Block ',i,' Seg ',PtrStruc(vptr[i]).seg:4,'  ',
                        ' Ofs ',PtrStruc(vptr[i]).ofs:4        );
       writeln('Block ',i+1,' Seg ',PtrStruc(vptr[i+1]).seg:4,'  ',
                        ' Ofs ',PtrStruc(vptr[i+1]).ofs:4     );
   end;
   if contiguous_memory then begin
     ALLOC_MEMORY:=vptr[1];
     Exit;
   end;
   for i:=1 to nBlocks-1 do
     MyFreeMem(vptr[i],BlockSize);
   MyFreeMem(vptr[nBlocks],lastBlockSize);
   ALLOC_MEMORY:=NIL;
 end;

procedure FREE_MEMORY( p:pointer; n:longint );

 var
     nBlocks,i:integer;
     lastBlockSize:word;
     size:longint;

 begin
   size:=(n+1)*Sizeof(elementtype);
   nBlocks:=(size-1) div BlockSize+1;
   lastBlockSize:=word(size-(nBlocks-1)*BlockSize);
   for i:=1 to nBlocks-1 do begin
     MyFreeMem(p,BlockSize);
     Inc(PtrStruc(p).seg,SegIncr);
   end;
   MyFreeMem(p,lastBlockSize);
 end;

function ith_position( base_ptr:pointer; i:ith_arr_element ) :
                                                 ptr_to_element;
 var
    segs_,ofs_:word;
    membytes:longint;

 begin
   membytes:=longint(i*Sizeof(elementtype));
   segs_:=word(membytes div BlockSize);
   ofs_:=word(membytes-segs_*BlockSize);
   Inc( PtrStruc(base_ptr).seg,segs_*SegIncr);
   Inc( PtrStruc(base_ptr).ofs,ofs_);
   ith_position:=ptr_to_element(base_ptr);
 end;

function ith_element( base_ptr:pointer; i:ith_arr_element ) :
                                                  elementtype;
 begin
   ith_element:=ptr_to_element(ith_position(base_ptr,i))^;
 end;

procedure ith__element( base_ptr:pointer; i:ith_arr_element;
                        var v_i:elementtype              );
 begin
   v_i:=ptr_to_element(ith_position(base_ptr,i))^;
 end;

procedure Wait_to_Go;

 var
     ch:char;

 begin
   write('Press any KEY to continue...');
   while not KeyPressed do ;
   ch:=ReadKey;
   if ch=#0 then ch:=ReadKey;
 end;

 begin
   HeapError:=@HeapFunc;
 end.
```

Wydruk 1. Jednostka LongRArr – definicje typów i implementacje operacji

Technika programowania jest tu podobna do techniki stosowanej w języku C. Używa się wskaźnika wektora (tablicy) elementów typu **elementtype**:

typedef float elementtype
....
elementtype far *base_ptr;

Pozycja *i*-tego elementu jest jednak wyznaczona jako

base_ptr+i

natomiast jego wartość jako

*(base_ptr+i)

```
program LongArray;

 uses mWindowU,LongRArr,CRT;

procedure Use_Long_Vector;

 var
     i : longint; r : integer;
     base_ptr : pointer;
     vW : UserWindows;
     ch : char;
     VideoMode : byte absolute $0040:$0049;
     n,size : longint;
     maxRow : integer;
     vm : VideoModeType;

 const
       first_window  : integer = 1;
       second_window : integer = 2;

 begin
   { Poniższe funkcje i procedury                                    }
   vm:=SET_VIDEO_MODE(VideoMode);
   INIT_STATUS_OF_WINDOWS( vW,3 );
   SET_TEXT_WINDOW( vW,first_window,'',1,1,last_Screen_Col,5,
                    30,2,SingleBrd,
                    LightMagenta,LightGray,
                    LightMagenta,Green,
                    ClearOn,DisplayOn                          );
   DISPLAY_TITLE( vW,first_window,'Large Vector',Centered);
   SET_TEXT_WINDOW( vW,second_window,'',1,6,
                    last_Screen_Col,last_Screen_Row-1,
                    1,1,DoubleBrd,
                    Yellow,Black,
                    Green,Red,
                    ClearOn,DisplayOn                          );
   { zawiera jednostka mWindowU                                      }

   maxRow:=HI(WindMax)-HI(WindMin);
   repeat
     write('How many real numbers? '); readln(n);
     if n<=0 then Exit;
     base_ptr:=ALLOC_MEMORY(n);
     if base_ptr=NIL then begin
       writeln('Memory not allocated!');
       Halt(1);
     end;
{    for i:=1 to n do begin
       write('number ',i:6,' is ' );
       readln(ith_position(base_ptr,i)^);
     end;                                                 }
     for i:=1 to n do
      ith_position(base_ptr,i)^:=i;
     r:=0;
     for i:=1 to n do begin
       write(ith_element(base_ptr,i):8:2,' ' );
       if i mod 7 =0 then begin
       writeln; r:=r+1;
       if r>=maxRow then begin
         Wait_to_Go; writeln;
         r:=0;
       end
     end;
   end;
   if i mod 7 <>0 then writeln;
   FREE_MEMORY(base_ptr,n);
   until false;
end;

 var
    vm:VideoModeType;

 begin
   Use_Long_Vector;
   Wait_to_Go;
   vm:=SET_VIDEO_MODE(LastMode);
   NormVideo;
end.
```

Wydruk 4. Program włączający plik definicji typów LongArr.typ i plik implementacji operacji

Na wydrukach 3 i 4 przedstawiono krótkie programy demonstrujące wykorzystanie długiego wektora liczb. Pierwszy program (wydruk 3) wykorzystuje jednostkę LongArr, drugi natomiast (wydruk 4) włącza pliki z definicjami typów i implementacją operacji dla długiego wektora. Przedstawiona koncepcja operowania długim wektorem może być wykorzystana w implementacji różnorodnych algorytmów wykorzystujących macierze oraz wektory. Rozmiar bloku BlockSize oraz przyrost części adresowej SegIncr można ustawić na inne wartości, np.:

```
BlockSize : word = $00F0;
SegIncr   : word = $000F;
```

LITERATURA

[1] Aho A. V., Hopcroft J. E., Ullmann J. E.: Data Structures and Algorithms. Addison-Wesley, Reading (MA), 1983

[2] Runka H. J.: Wstęp do teorii algorytmów. Struktury danych w projektowaniu algorytmów na grafach. Wydawnictwa AE Poznań. Skrypt uczelniany 405 (SU 405)

[3] Turbo Pascal Reference Guide. Version 5.0, Borland International, 1988.

Możliwości stosowania światłowodów...

dokończenie ze s. 2

Wyniki przeprowadzonych badań konwertera KS-1 świadczą, że może on być stosowany do przesyłania informacji w systemach abonenckich, w których informacja jest przesyłana w relacji dwupunktowej, a także w niektórych lokalnych sieciach komputerowych, np. (w sieci LAN-Link). W Wojskowym Instytucie Informatyki pracuje się nad budową konwerterów, w których pasmo przenoszenia traktu światłowodowego byłoby dzielone na podkanały umożliwiające bezkolizyjne przesyłanie tym samym kablem światłowodowym trzech podstawowych sygnałów: zajętości, danych i sygnału zegara. Konwertery te będą również dostosowane do współpracy z innymi niż RS 232C sprzęgami, co umożliwi budowę magistrali światłowodowych dla innych typów lokalnych sieci komputerowych.

* * *

Koszt materiałów zużytych do budowy jednego konwertera KS-1 wynosił według cen z 1988 r. ok. 50 000 zł. Do połączenia konwerterów zastosowano jednożyłowy kabel światłowodowy $LWLK_V$ 1×1 S. Ze względu na to, że moduły nadajnika i odbiornika optoelektrycznego są zakończone półłączami rozłącznymi, kabel ten powinien być również zakończony takimi półłączami. W tym celu zakupiono kabel TY6200002 (produkcji CSRS) o długości 2 m. Po przecięciu tego kabla w połowie długości, obydwa odcinki zostały przyspawane do odpowiednio długiego odcinka kabla $LWIK_V$ 1×1 (operacja spawania światłowodów może być zastąpiona ich klejeniem). Można również zakupić potrzebne odcinki kabla światłowodowego produkcji CSRS typu TY6200XXX zakończone półłączami (o długości do 1 km). Zamawianą długość w metrach podaje się na pozycjach XXX w oznaczeniu kabla. Zarówno kable światłowodowe, jak i układy WK-164-41 i WK-164-42 można zakupić w PHZ Unitra. Wojskowe Centralne Biuro Konstrukcyjno-Technologiczne w Warszawie zamierza rozpocząć produkcję omawianych konwerterów światłowodowych.

Zanim podejmiesz decyzję o imporcie

z a p r e n u m e r u j

PRZEGLĄD POLSKIEJ TECHNIKI

Dwumiesięcznik, wydawany w językach: angielskim, niemieckim i rosyjskim, szczegółowo informuje o najnowszych osiągnięciach polskiego przemysłu i rodzimej myśli technicznej.

Cena prenumeraty rocznej wynosi 92 125 zł.

Zgłoszenia prenumeraty – Dział Kolportażu Wydawnictwa SIGMA-NOT, ul. Bartycka 20, 00–716 Warszawa, tel. 40 30 86 lub 40 00 21 w. 293

MARIUSZ ORMIŃSKI
Computer Studio Kajkowski
Sopot

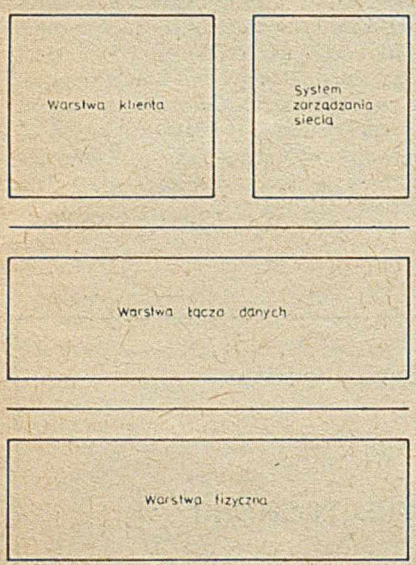
Ethernet – od specyfikacji do realizacji praktycznej (1)

Specyfikacja

Pierwsze prace nad lokalną siecią komputerową Ethernet, rozpoczęto w firmie Xerox. Zaowocowały one powstaniem w 1975 roku tzw. eksperymentalnej sieci Ethernet. Połączenie wysiłków firm DEC Intel i Xerox doprowadziło do opracowania i opublikowania we wrześniu 1980 roku pierwszej wersji specyfikacji tej sieci [1], a następnie w listopadzie 1982 roku kolejnej wersji specyfikacji [2]. W 1985 roku opublikowano normę IEEE 802.3 [3] specyfikującą całą klasę sieci typu CSMA/CD. W ramach tej normy specjalnie wyróżniono rozwiązanie nieomal identyczne z przedstawionym w specyfikacji firm DEC Intel i Xerox.

ARCHITEKTURA SIECI ETHERNET

Architekturę protokołu sieci Ethernet utworzono zgodnie z zalecanym przez ISO siedmiowarstwowym modelem OSI (ang. *open systems interconnection*), przy czym specyfikacja objęła tylko dwie najniższe warstwy – fizyczną i łącza danych. Pozostałe warstwy bez wnikania w ich funkcje i strukturę zostały potraktowane całościowo jako użytkownik usług warstwy łącza danych i nazwane, na potrzeby specyfikacji, warstwą klienta (ang. *client layer*). Dodatkowo, do celów kontrolnych i sprawnego sterowania pracą stacji w sieci, wyróżniono specjalnego użytkownika usług warstwy łącza danych, zwanego systemem zarządzania siecią (ang. *network management system*), dla którego przewidziano dodatkowy sprzęg z warstwą łącza danych. Architekturę protokołu sieci typu Ethernet przedstawiono na rys. 1.



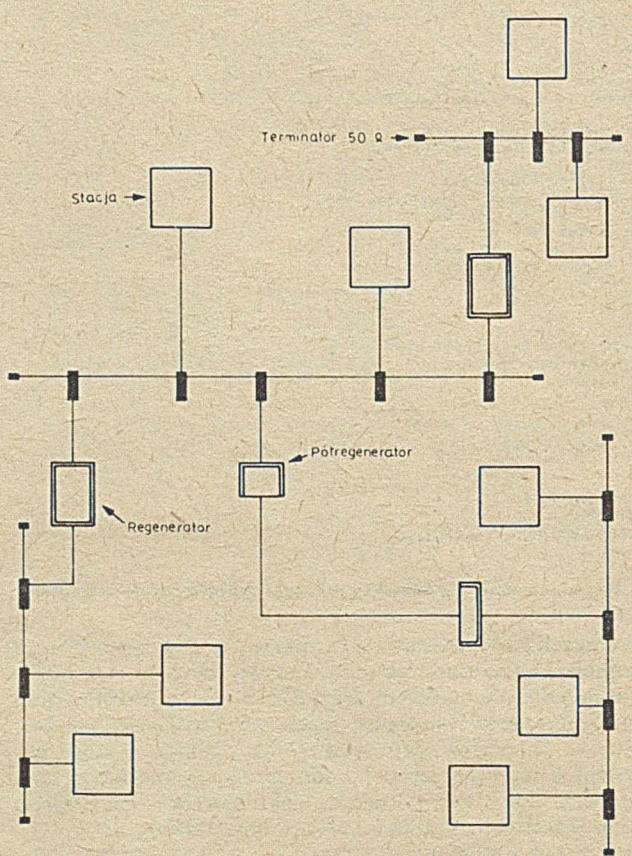
Rys. 1. Warstwowa struktura protokołu sieci typu Ethernet

Odnośnie warstwy łącza danych należy dodać, że mimo tej nazwy faktycznie specyfikacją są objęte tylko pewne jej funkcje. Obejmuje ona jedynie funkcje podwarstwy sterowania dostępem do kanału komunikacyjnego. Pozostałe z realizowanych przez tę warstwę funkcji przeniesiono do warstwy klienta. Są to funkcje drugiej podwarstwy przeznaczonej do sterowania połączeniami logicznymi.

Sieć Ethernet jest siecią o konfiguracji magistralowej. Wszystkie urządzenia są dołączone do kabla współosiowego o impedancji 50 Ω. Dostęp do kanału komunikacyjnego odbywa się według reguły CSMA/CD (ang. *carrier sense, multiple access with collision detection*), czyli jest to dostęp do wspólnego kanału z wykrywaniem nośnej i kolizji. Szybkość transmisji w kanale wynosi 10 Mb/s, maksymalna liczba komunikujących się stacji wynosi 1024, a odległość między dwiema najdalszymi stacjami nie może przekroczyć 2800 m.

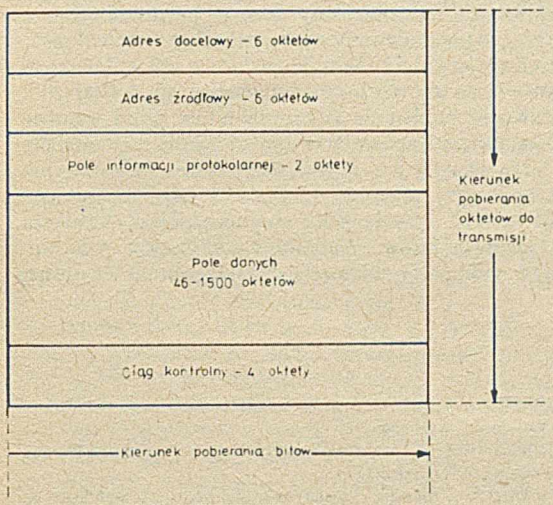
Ponieważ nie zawsze można usytuować stacje bezpośrednio przy kablu współosiowym, dołączenie do sieci wykonuje się przez wyodrębnione ze sterownika dodatkowe urządzenie nadawczo-odbiorcze (ang. *transceiver*). Urządzenie to jest odpowiedzialne za nadawanie i odbiór poszczególnych ramek, badanie zajętości kanału i wykrywanie kolizji. Może się zdarzyć, że potencjały mas różnych stacji będą różne, aby więc zapobiec przepływom dużych prądów, jest ono odizolowane galwanicznie od stacji. Stację łączy się z tym urządzeniem za pomocą kabla o długości nie przekraczającej 50 m.

Chociaż maksymalna odległość między dwiema stacjami wynosi 2800 m, jednak bezpośrednie połączenie między nimi nie może przekroczyć 500 m. Tyle też wynosi maksymalna długość jednego segmentu kabla współosiowego. Do jednego segmentu nie można dołączyć więcej niż 100 stacji. W celu zwiększenia liczby łączonych stacji oraz zasięgu sieci, można stosować regeneratory (ang. *repeater*) umożliwiające łączenie segmentów ze sobą. Na drodze między dwiema stacjami mogą wystąpić najwyżej dwa regeneratory. Regenerator może zostać rozdzielony na dwie jednostki połączone kablem, zwane półregeneratorami (ang. *half-repeater*). Sumaryczna długość kabla łączącego pary półregeneratorów nie może przekroczyć 1000 m. Stosując regeneratory należy unikać zamkniętych pętli. Przykładowe rozwiązanie struktury sieci przedstawiono na rys. 2.



Rys. 2. Przykładowa konfiguracja sieci typu Ethernet

Najmniejszą jednostką informacji przesyłaną między stacjami jest ramka (ang. *frame*). Składa się ona z następujących pól: pola adresu docelowego, pola adresu źródłowego, pola informacji protokolarnej warstw wyższych, pola danych i pola kontrolnego (rys. 3). Pola adresowe mają długość 6 oktetów (48 bitów). Adres może być adresem fizycznym odwołującym się do jednej konkretnej stacji lub adresem wielokrotnym, a więc adresem wielu stacji. Adres wielokrotny może być adresem grupowym, wtedy dotyczy pewnej grupy stacji, lub może oznaczać, że ramka w trybie rozgłaszania jest kierowana do wszystkich stacji. Adresy wielokrotne mogą wystąpić tylko w polu adresu docelowego. Adres fizyczny jest liczbą parzystą, a wielokrotny – nieparzystą.



Rys. 3. Struktura ramki w sieci typu Ethernet

Pole informacji protokolarnej warstw wyższych ma długość dwóch oktetów i jest przeznaczone do przekazywania dodatkowych informacji protokolarnych. Jest ono interpretowane na poziomie warstw wyższych niż warstwa łącza danych. Pole danych jest polem o zmiennej długości – od 46 do 1500 oktetów. Ograniczenie długości pola danych (a więc i ramki) od góry jest spowodowane ograniczeniem czasu oczekiwania na niezajęty kanał, a także ograniczonymi pojemnościami buforów w stacji lub w samym sterowniku. Ograniczenie od dołu jest wynikiem dążenia do uzyskania zadowalającej efektywności wykrywania kolizji. Krytycznym paramatrem jest tu czas propagacji determinujący minimalny czas potrzebny na wykrycie kolizji. W razie kolizji, sterownik zaprzestaje nadawania i odbiornik może wykryć ten stan odbierając ramkę krótszą niż dopuszczalna. Gdyby nie było ograniczenia długości ramki od dołu, to mimo kolizji ramki takie mogłyby być potraktowane przez warstwę łącza danych jako poprawne i kolizja byłaby wykrywana dopiero przez warstwy wyższe.

Ostatnim polem ramki jest pole kontrolne o długości 4 oktetów. Do wykrywania błędów transmisji zastosowano cykliczny kod nadmiarowy (CRC). Wartość wpisywana do tego pola jest obliczana na podstawie wielomianu generującego $G(x)$ postaci:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Jak można zauważyć, w ramce nie występuje znacznik jej końca, ani nie ma pola informującego o jej długości. Koniec ramki jest wykrywany przez badanie momentu zaniku nośnej.

Podczas nadawania ramek obowiązuje zasada, że oktet jest wysyłany począwszy od bitu o najmniej znaczącej pozycji. Bezpośrednio przed nadaniem ramki jest wysyłany 64-bitowy ciąg synchronizujący, zwany **preambułą.** Preambuła składa się z występujących na przemian jedynek i zer, z wyjątkiem dwóch ostatnich bitów, które są jedynkami. Preambuła jest wysyłana w celu ułatwienia synchronizacji odbioru zarówno na pozomie bitów, jak i oktetów oraz całej ramki. Aby umożliwić sterownikom odczekanie do chwili zaniknięcia stanów przejściowych, przyjęto, że odstęp między kolejnymi ramkami musi być nie mniejszy niż 9,6 μs. W celu przesłania ciągu bitów przez kanał zastosowano w sieci Ethernet kod Manchester, ze względu na jego prostotę oraz właściwości ułatwiające synchronizację.

NADAWANIE I ODBIÓR RAMEK

Procesy nadawania i odbioru ramek w sieci Ethernet, realizowane przez warstwę fizyczną i łącza danych, są ściśle związane z przyjętą regułą dostępu do kanału komunikacyjnego. Stanowią więc charakterystyczną cechę tego typu sieci.

Nadawanie jest incjowane przez warstwę klienta, która przekazuje odpowiednie informacje i dane. Na ich podstawie oraz na podstawie wyznaczonego ciągu CRC sterownik tworzy ramkę. Jeżeli kanał komunikacyjny jest zajęty, to sterownik czeka na zanik nośnej. Po odczekaniu 9,6 μs na zanik stanów przejściowych przystępuje do nadawania preambuły, a potem ramki. Sterownik może przystąpić do transmisji natychmiast, jeśli od chwili zaniku nośnej w kanale upłynął czas minimalnego odstępu międzyramkowego. Jeśli po rozpoczęciu transmisji sterownik wykryje kolizję, czyli jednoczesną próbę nadawania przez inne stacje, to sygnalizuje ten fakt przez tzw. zagłuszanie (ang. *jamming, jam*), tj. wysyłanie losowego ciągu o długości od 32 do 48 bitów. Po kolizji odczekuje pewien czas i ponownie przystępuje do nadawania. Czas odczekiwania jest losową wielokrotnością przedziału kolizji wybraną z zakresu $\langle 0,2^k)$, przy czym $k = \min(n, 10)$, a n określa numer kolejnej próby retransmisji. Przedział kolizji jest przedziałem czasowym zdeterminowanym przez czas propagacji sygnałów w kanale. Wszystkie kolizje przy normalnych warunkach pracy powinny wystąpić w ramach tego przedziału. Może się jednak zdarzyć, że z powodu awarii któregoś ze sterowników kolizje będą występowały później. Maksymalna liczba wszystkich prób nadawania, czyli pierwszej próby oraz retransmisji, wynosi 16. Nadawanie jest kończone przekazaniem do warstwy klienta informacji o rezultacie transmisji.

Odbiór jest inicjowany przez warstwę klienta. Pojawienie się nośnej w kanale komunikacyjnam powoduje, że sterownik, na podstawie preambuły, dokonuje synchronizacji i po jej zakończeniu przystępuje do odbioru ramki. Odbiór trwa tak długo, jak długo jest obecna nośna. Po zakończeniu odbioru bada się długość ramki i jeśli jest ona zbyt mała, to ramka zostaje odrzucona (jest to bowiem z reguły wynik wystąpienia kolizji). Następnie bada się pole adresu docelowego, aby ustalić, czy ramka ma być odebrana przez stację. Jeśli tak jest, to bada się ciąg kontrolny i właściwą długość ramki. Odebrane dane są następnie przekazywane warstwie klienta wraz z dodatkową informacją o ewentualnych błędach.

FUNKCJE DOLNYCH WARSTW

Warstwa fizyczna realizuje dwie główne funkcje ściśle związane z kanałem komunikacyjnym tj.:

- synchronizację i kodowanie, polegające na generowaniu i usuwaniu preambuły oraz kodowaniu i dekodowaniu ciągu bitów,
- dostęp do kanału, sprowadzający się do nadawania i odbioru zakodowanego ciągu bitów, wykrywania nośnej w kanale i wykrywania kolizji.

Funkcję dostępu do kanału realizuje urządzenie nadawczo-odbiorcze.

Warstwa łącza danych realizuje funkcje komunikacyjne na poziomie niezależnym od ośrodka komunikacyjnego:

- zarządzanie jednostkami informacji, na co składa się tworzenie ramki, generowanie i wykrywanie adresów, wykrywanie błędów transmisji,
- zarządzanie komunikacją, polegające na sterowaniu dostępem do kanału i obsłudze kolizji.

Warstwa ta jest więc odpowiedzialna za utworzenie ramki przy nadawaniu, pobieranie informacji i danych z ramki w czasie odbioru, sprawdzanie adresów, generowanie ciągu CRC i wykrywanie błędów na jego podstawie, a także za sterowanie komunikacją na podstawie otrzymanych z warstwy fizycznej sygnałów obecności nośnej i wystąpienia kolizji.

System zarządzania siecią dostarcza usług koniecznych do zarządzania i sterowania pracą sieci. Nie wchodzi on w skład sterownika i jest realizowany w ramach oprogramowania sterującego pracą sterownika. Z uwagi na pełnione przez siebie funkcje może komunikować się nie tylko z warstwą łącza danych, ale również z wyższymi warstwami sieci. Problem ten wykracza jednak poza granice specyfikacji sieci typu Ethernet. W stosunku do warstwy łącza danych system zarządzania siecią realizuje takie funkcje, jak:

- sterowanie pracą sterownika przez:

– inicjowanie, zawieszanie i wznawianie działania warstwy łącza danych,

– ustawianie fizycznego adresu stacji,
– ustawianie trybu adresowania stacji (normalny, wielokrotny tryb odbioru wszystkich ramek),
● kontrola działania, tj.:
– śledzenie wartości parametrów warstwy łącza danych,
– statystyka pracy sterownika,
– statystyka błędów.

Ustawianie trybu adresowania stacji umożliwia odbiór ramek nie tylko na własny adres fizyczny, ale także na adres grupowy. Można również wybrać taki tryb pracy, w którym będą odbierane wszystkie ramki pojawiające się w kanale komunikacyjnam. Funkcja kontroli działania umożliwia bieżące śledzenie pracy sterownika. Podczas jej realizacji warstwa łącza danych przekazuje informacje o liczbie poprawnie wysłanych i odebranych ramek, liczbie ramek nie wysłanych z powodu wielokrotnych kolizji, liczbie kolizji, które wystąpiły poza przedziałem kolizji, oraz – liczbie odebranych ramek mających nieprawidłową długość lub zawierających błąd CRC. Przekazuje się również informację o niewłaściwym wykrywaniu nośnej i kolizji.

SPRZĘGI MIĘDZYWARSTWOWE

Warstwa łącza danych przez sprzęg z warstwą klienta świadczy jej usługi nadawania i odbierania ramek. Korzystając z tego sprzęgu, warstwa klienta przekazuje wszystkie niezbędne informacje i dane potrzebne do wysłania ramki oraz pobiera analogiczne informacje i dane dotyczące ramki odebranej. Są to: adres docelowy, adres źródłowy, informacja protokolarna warstw wyższych oraz ciąg bajtów wypełniających pole danych. Dodatkowo, po zakończeniu nadawania lub odbioru ramki, warstwa klienta otrzymuje informacje o rezultacie wykonanej operacji. Odbiór ramki, podobnie jak nadawanie, musi zostać zainicjowany przez warstwę klienta. Jeśli operacja ta nie zostanie zainicjowana, to sterownik nie będzie odbierał ramki.

Proces nadawania może zakończyć się:
– transmisją poprawną bez kolizji,
– transmisją poprawną z jedną kolizją,
– transmisją poprawną po wielokrotnej kolizji,
– transmisją niepoprawną w wyniku przekroczenia maksymalnej liczby prób,
– błędem transmisji spowodowanym kolizją poza przedziałem kolizji,
– sterownikiem wyłączonym.

Z kolei rezultat odbioru może być następujący:
– odbiór poprawny
– ramka odebrana z błędami,
– ramka odebrana o niewłaściwej długości (niepełny ostatni oktet),
– sterownik wyłączony.

Przez sprzęg z warstwą fizyczną warstwa łącza danych przekazuje jej ciąg bitów przeznaczonych do nadawania oraz pobiera ciąg bitów odebranych. Korzysta również z dostarczonej przez warstwę fizyczną procedury odczekiwania przez określony czas, tj. przeterminowania (ang. *time-out*). Czas ten jest podawany w jednostkach, których długość zależy od fizycznych właściwości kanału. Procedura ta jest wykorzystywana przez warstwę łącza danych przy odczekiwaniu po wystąpieniu kolizji. Przy nadawaniu pobieranie kolejnych bitów przez warstwę fizyczną jest sterowane przez warstwę łącza danych. W razie wystąpienia kolizji warstwa łącza danych jest o tym natychmiast informowana i przekazywanie bitów jest przerywane. Podczas odbioru warstwa fizyczna informuje warstwę łącza danych o obecności nośnej i wymusza na niej odbiór kolejnych bitów. Warstwa łącza danych przerywa pobieranie bitów w momencie zniknięcia sygnału obecności nośnej w kanale.

LITERATURA

[1] DEC, Intel, Xerox: The Ethernet – A Local Area Network. Data Link Layer and Physical Layer Specyfications. Version 1.0, September 1980
[2] DEC. Intel, Xerox: The Ethernet – A Local Area Network. Data Link Layer and Physical Layer Specifications. Version 2.0, November 1982
[3] IEEE: Standards for Local Area Networks. Carrier Sense Multiple Access with Collision Detection (CSMA/CD) – Access Method and Physical Layer Specifications. ANSI/IEEE Std 802.3, 1985
[4] Sòch J.F., Dalah Y.K. Redell D.D., Crane R.C.: The Ethernet. Pp. 1–35, Proc. Conf. Local Area Networks – An Advanced Course, Glasgow. 11–12 July 1983.

digitex

ZAKŁAD SYSTEMÓW CYFROWYCH
81-832 Sopot, ul. Mickiewicza 20
tel. 51-28-27, tlx. 512290 dgtex pl

KOLOROWY TERMINAL GRAFICZNY DXT–240

● emuluje protokół terminali graficznych **DEC-a VT–240, VT–125 i TEKTRONIX 4014**
● emuluje protokół terminali alfanumerycznych DEC-a VT–220, VT–100, VT–52
● przeznaczony do pracy w konfiguracjach wielodostępnych z komputerami serii VAX, SM, PDP11, PC XT/AT pod kontrolą systemów operacyjnych XENIX, UNIX, RSX, MULTILINK i innych
● wyświetla informacje nie tylko w postaci znaków alfanumerycznych, ale również w postaci wykresów, rysunków itp., zgodnie z protokołem graficznym **ReGIS**, który m.in. zapewnia kreślenie prostych, okręgów, elips, krzywych, zapełnianie, „dostęp" do kazdego punktu itp.
● rozdzielczość grafiki:
640 × 256 punktów – DXT–240A
640 × 400 " – DXT–240B
640 × 480 " – DXT–240C
800 × 480 " – DXT–240D
● atrybuty: 16 kolorów kazdego punktu (lub 8 kolorów i migotanie)
● tryb alfanumeryczny: max. 60 wierszy po 132 znaki
● interface komunikacyjny RS 232 lub pętla prądowa
● interface drukarki Centronix lub RS 232
● klawiatura typu PC–XT
● monitor kolorowy 14" (RGB standard, EGA, Multisync – zależnie od rozdzielczości terminala) lub monitor monochromatyczny.
☆ DEC, VT, ReGIS są zastrzeżonymi znakami firmowymi Digital Equipment Co.
☆ TEKTRONIX jest zastrzeżonym znakiem firmy Tektronix Inc.

EO/646/89

MIECZYSŁAW KŁOPOTEK, ADAM KOWALSKI, ROBERT MACURA, MACIEJ MICHALEWICZ, ANDRZEJ PACAN, ELŻBIETA SYROPIATKO, SŁAWOMIR WIERZCHOŃ
Instytut Podstaw Informatyki PAN
Warszawa

SYS8688 – system pozyskiwania i weryfikacji wiedzy medycznej (1)

Próby formalizacji procesów decyzyjnych w medycynie, polegające na budowie systemów informacyjnych modelujących podejmowanie decyzji przez specjalistów są prowadzone od dawna. Można wyróżnić trzy zasadnicze kategorie takich systemów:

- systemy oparte na analizie statystycznej danych medycznych,
- systemy wykorzystujące metody rozpoznawania obrazów,
- medyczne systemy ekspertowe (SE).

W systemach dwóch pierwszych kategorii był wykorzystywany formalny aparat matematyczny, SE natomiast – powstałe jako ich alternatywa – są oparte na przetwarzaniu informacji przedstawionej w postaci symbolicznej i reprezentującej wiedzę eksperta [1]. Główną przyczyną pojawienia się SE była niemożność interpretacji w języku medycznym sposobu, w jaki pozostałe systemy osiągają rozwiązanie. Modele matematyczne, opracowane głównie dla nauk ścisłych, okazały się mało przydatne, gdyż wiedza o dziedzinie opisywanej przez taki model (semantyka modelu) pozostawała na zewnątrz komputera, w umyśle człowieka rozwiązującego zadanie. Odmienność podejścia zastosowanego w SE polega na wykorzystaniu modeli logiczno-lingwistycznych, opracowanych w ramach sztucznej inteligencji, wyrażanych w języku dziedziny zastosowania, a więc w pełni zrozumiałych dla większości specjalistów danej dziedziny.

Wyróżnione kategorie systemów mają ponadto wiele innych ograniczeń, stanowiących istotną przeszkodę w ich upowszechnieniu w środowiskach pozaakademickich (poliklinikach, oddziałach szpitalnych itd.). Wadą systemów statystycznych jest to, że użytkownik sam musi formułować hipotezy, oraz że proces wnioskowania statystycznego wykonywany przez system w trybie „off-line" jest w praktyce także w rękach użytkownika. Systemy te sprawdzają wyłącznie formalne spełnienie założeń stosowanych testów, co implikuje konieczność znajomości statystyki lub zasięgnięcia rady specjalisty. Wadą systemów drugiej kategorii jest niejawność procesu wnioskowania (system działa na zasadzie automatu diagnozującego). Podejście takie jest uzasadnione wyłącznie w dobrze poznanych obszarach medycyny.

Istotną przeszkodą w rozwoju SE jest to, co z początku wydawało się proste i oczywiste: pozyskiwanie wiedzy. Mimo poważnych badań nie osiągnięto w tej dziedzinie znaczących postępów i w dalszym ciągu gromadzona wiedza ekspertów bywa niepełna, subiektywna i często po prostu sprzeczna. Pojawienie się SE stanowi jednak istotny postęp, gdyż – w odróżnieniu od systemów baz danych – umożliwiają one aktywne operowanie wiedzą.

Nietrudno zauważyć, że każde z omówionych podejść daje tym lepsze wyniki, im lepiej znany jest badany proces. Można, na przykład, za pomocą SE automatycznie przeanalizować sygnał EKG i postawić odpowiednią diagnozę. Warto jednak przy tym pamiętać, że dobry kardiolog bez żadnych trudności osiąga równie dobre (lub lepsze) wyniki, gdyż dziedzina ta jest dość dobrze poznana.

Największe trudności sprawiają te obszary medycyny, które są jeszcze stosunkowo słabo poznane. Trudno w nich o hipotezy, wiedza jest fragmentaryczna i niepełna, dostępne są natomiast dane o poszczególnych przypadkach chorobowych. W takiej sytuacji każda z wymienionych kategorii systemów całkowicie zawodzi. Powodem takiego stanu rzeczy – oprócz wspomnianych wyżej ograniczeń – jest to, że – żaden z systemów nie pozwala na równoczesne operowanie danymi i wiedzą ani na statystyczną weryfikację generowanych hipotez. Potrzebne byłoby więc narzędzie, które umożliwiłoby:

– zapis zbioru danych (w tym – charakterystyk czasowych danych),
– zapis istniejącej wiedzy,
– analizę statystyczną wybranych zbiorów danych bez konieczności znajomości statystyki, np. w postaci prośby o odpowiedź na konkretne pytanie sformułowanie w języku medycyny.
– automatyczne generowanie hipotez bezpośrednio ze zbioru danych,
– powiązanie wiedzy z danymi (w szczególności formułowanie pytań o występowaniu związku między porcją wiedzy a daną hipotezą oraz wyszukiwanie odpowiednich podzbiorów danych z jednoczesną weryfikacją statystyczną danej hipotezy).

ZAŁOŻENIA KONCEPCYJNE I ARCHITEKTURA SYS8688

Opracowany przez autorów artykułu ekspertowo–statystyczny system SYS8688 jest próbą wyjścia naprzeciw sformułowanym powyżej wymaganiom. Jest on przeznaczony do prowadzenia badań naukowych, w wybranych, słabo poznanych obszarach medycyny, tworzenia bazy wiedzy (BW) dotyczącej tej dziedziny i późniejszego jej wykorzystania w procesie konsultowania indywidualnych pacjentów.

SYS8688 może działać w dwóch trybach:

- w trybie badawczym, w którym jest rozszerzany zasób wiedzy systemu przez badania statystyczne oraz eksperymenty konsultacyjne,
- w trybie użytkowym, w którym system udziela konsultacji, gromadzi dane itp.

System jest przeznaczony dla placówek naukowo–badawczych oraz dla klinik typowych szpitali, w których może pełnić nie tylko funkcje badawcze, lecz również przyczynić się do szybkiego rozpowszechniania najnowszej wiedzy medycznej, w postaci „aktywnego konsultanta".

Na SYS8688 składają się systemy:

– bazy danych,
– bazy wiedzy,
– ekspertowy.
– pozyskiwania wiedzy.

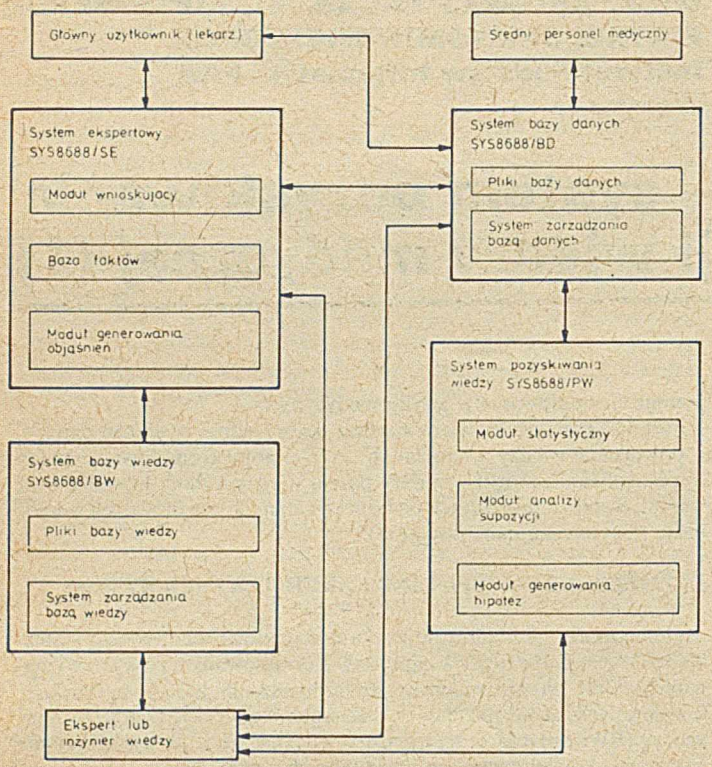
Architekturę i kategorie potencjalnych użytkowników SYS8688 przedstawiono na rysunku.

Przedstwiona struktura systemu wspomagania badań naukowych jest ilustracją jednej z bardziej znanych definicji nauki, w myśl której nauka – to tworzenie teoretycznych uogólnień, wynikających z obserwacji oraz doświadczeń, i prowadzących do nowych obserwacji i nowych doświadczeń. Obserwacje i doświadczenia to fakty zgromadzone w bazie danych (BD). Teoretyczne uogólnienia to reguły wnioskowania zawarte w BW. Postęp w pracach badawczych jest możliwy dopiero wówczas, gdy nastąpi sprzężenie zwrotne między tymi dwoma rodzajami informacji: system pozyskiwania wiedzy umożliwia dokonywanie uogólnień na podstawie danych z BD, tworząc gotowe reguły wnioskowania dla BW, a SE umożliwia wykorzystanie teoretycznych uogólnień do diagnozowania i określenia terapii dla konkretnego pacjenta, który z kolei staje się jakościowo nowym obiektem obserwacji.

W przypadku medycyny materiał badawczy jest bogaty (z uwagi na jego liczebność i różnorodność). W celu ustalenia ostatecznego rozpoznania klinicznego i terapii bądź w celu wykrycia nowych elementów różnicujących choroby niezbędne jest zestawienie jednych danych (parametrów) z innymi, a następnie całościowe ujęcie informacji o chorym. Ogrom informacji powoduje jej pasywność, jest też przyczyną niepełnej i nieefektywnej analizy danych przez człowieka. SYS8688 zapewnia pełny dostęp do danych (informacje o chorych) i do wiedzy (np. charakterystyki chorób) dzięki mechanizmowi kwerend do BD i BW oraz dzięki modułom analizy i pozyskiwania wiedzy. Ułatwia wyszukiwanie precedensów oraz usprawnia proces uogólniania. System zawiera wreszcie wiele mechanizmów wspomagających procesy formułowania i weryfikacji hipotez badawczych.

Cechą charakterystyczą SYS8688 jest nie tyle wspólna implementacja SE oraz systemów BD, BW i pozyskiwania wiedzy, co ich połączenie

w narzędzie pracy nowego typu, oparte na jednolitych formatach danych i na jednolitych metodach dostępu do danych.



Architektura systemu SYS8688

SYSTEM BAZY WIEDZY

System bazy wiedzy składa się z:
- systemu zarządzania bazą wiedzy (SZBW),
- plików bazy wiedzy.

Do podstawowych funkcji systemu należy gromadzenie wiedzy eksperta o danej dziedzinie oraz zapewnienie innym modułom efektywnego dostępu do tej wiedzy.

System zarządzania bazą wiedzy

SZBW realizuje wiele funkcji związanych z obsługą BW. Na szczególną uwagę zasługują:
- mechanizm kwerend do BW,
- stemplowanie reguł,
- redagowanie BW sterowane składnią.

Mechanizm kwerend odgrywa bardzo ważną rolę w gromadzeniu i weryfikacji wiedzy. Formułując kwerendy w postaci zbioru atrybutów i (lub) konkluzji, ekspert – lub współpracujący z nim inżynier wiedzy – testuje BW, wyszukując miejsce wymagające uzupełnień bądź modyfikacji.

Stemplowanie reguł ma na celu zapamiętanie wszelkich zmian dokonywanych w zbiorze reguł przez ekspertów. Wykorzystując numer reguły jako klucz, SZBW wyszukuje jeden lub kilka rekordów, składających się z następujących pól:

⟨*identyfikator_eksperta*⟩⟨*data_akcji*⟩⟨*rodzaj_akcji*⟩

Akcje wykonywane na regułach to ich tworzenie, modyfikowanie, upasywnienie lub uaktywnienię.

Ważnym narzędziem pozyskiwania wiedzy, zapewniającym poprawność syntaktyczną reguł, jest edytor sterowany składnią reguł. Umożliwia on wykonywanie wspomnianych powyżej akcji na regułach oraz nie dopuszcza do wystąpienia w zbiorze reguł sprzeczności między współczynnikami pewności reguł [5].

Pliki bazy wiedzy

W skład bazy wiedzy SYS8688 wchodzą:
1) plik atrybutów (parametry kliniczne i laboratoryjne),
2) plik konkluzji (choroby, zaburzenia, prognozy),
3) plik reguł wnioskowania.

Plik atrybutów zawiera specyfikacje atrybutów postaci:

⟨*nazwa*⟩⟨*typ*⟩⟨*rodzaj*⟩⟨*źródło*⟩⟨ *zależność_czasowa*⟩{⟨*dziedzina*⟩}

Dopuszcza się stosowanie czterech typów atrybutów w zależności od przyjmowanych wartości:

- całkowitoliczbowy *C*,
- rzeczywisty *R*,
- tekstowy *T*,
- dyskretyzowany *D*.

Pole ⟨*rodzaj*⟩ określa, czy atrybut jest jedno – *J* czy też wielowarstwowy *W*. Atrybut rodzaju *J* (np. „płeć") może przybrać tylko jedną wartość ze zbioru wzajemnie wykluczających się wartości (kobieta, mężczyzna). Atrybut rodzaju *W* (np. „miejsce oparzenia") może mieć jednocześnie kilka wartości, które nie wykluczają się wzajemnie ({*twarz, szyja, kark...*}).

Istnieje pewna grupa atrybutów, których wartości mogą być ustalone wyłącznie w rezultacie odpowiedzi na pytanie postawione użytkownikowi systemu (ewentualnie w rezultacie pobrania ich wartości z BD). Najczęściej są to wyniki pomiarów lub badań laboratoryjnych. Informuje o tym pole ⟨*źródło*⟩, które może mieć wartości:
P – wartość atrybutu jest wyznaczona na podstawie odpowiedzi na zadane pytanie.
D – wartość atrybutu można wydedukować.

Pole ⟨*zależność_czasowa*⟩ określa, czy dany atrybut jest zmienny w czasie czy też nie. Dla atrybutów zależnych od czasu pole to określa dokładność, z jaką należy podawać punkty czasowe:
D – dokładność dobowa,
G – godzinowa,
M – minutowa.

Dziedzina przyjmuje różną postać w zależności od typu atrybutu. Dla atrybutów liczbowych *C, R* będzie to przedział, dla tekstowych *T* – lista możliwych wartości, a dla dyskretyzowanych *D* – lista par (przedział, interpretacja tekstowa). Sposób specyfikowania atrybutów można zilustrować na przykładzie parametru *„temperatura"*:

temperatura D J D G { [34.0, 36.4) *obniżona,*
[36.4, 37.0) *normalna,*
[37.0, 38.0) *podwyższona,*
[38.0, 42.0] *wysoka* }.

Plik konkluzji zawiera szczegółowy opis konkluzji. Zbiór konkluzji stanowi przestrzeń rozwiązań, w której operuje SE. Z każdą konkluzją jest związana lista reguł wnioskowania, które prowadzą do danej konkluzji, oraz tzw. sugestia terapeutyczna, określająca typowy sposób postępowania dla danej choroby lub zaburzenia (konieczność wykonania dodatkowych badań, konsultacji ze specjalistą, zestaw najskuteczniejszych leków itd.).

Reguły wnioskowania są przechowywane w pliku trzecim. W odróżnieniu od istniejących SE plik reguł w SYS8688 nie jest umieszczany w pamięci RAM. Zastosowano tu rozwiązanie polegające na zamianie pliku reguł na plik bezpośredniego dostępu z przechowywanym w pamięci RAM indeksem reguł, co zapewnia efektywne ich wyszukiwanie. Podejście to eliminuje ograniczenie dotyczące maksymalnej liczby reguł w BW, które poważnie utrudnia praktyczne zastosowania.

Schemat reprezentacji wiedzy

Jedną z kluczowych decyzji w procesie projektowania SE jest wybór schematu reprezentacji wiedzy. Podstawowe kryteria brane pod uwagę na tym etapie to rodzaj zadań rozwiązywanych przez system oraz struktura wiedzy danej dziedziny. Analizując medycynę w kontekście tych kryteriów, należy zauważyć, że:
- przeważająca część danych medycznych ma charakter jakościowy, a przy danych ilościowych rzadko występują zależności funkcyjne,
- związki przyczynowo-skutkowe między obserwacjami a hipotezami podlegają stałym modyfikacjom i uzpełnieniom.

Z tego powodu w SYS8688 zastosowano formalizm reguł, którego podstawową zaletą jest brak jawnego powiązania między porcjami wiedzy (regułami). Ma to ogromne znaczenie ze względu na łatwość modyfikowania i uzupełniania BW systemu [2].

Reguły wnioskowania mają postać zdań warunkowych:

IF ⟨*przesłanka*⟩ **THEN** ⟨*akcja*⟩

Przesłanka reguły składa się z koniunkcji warunków (porównań), przy czym każdy z nich może być z kolei alternatywą warunków. Warunki mogą być dwóch typów – uniwersalne lub czasowe, w zależności od typu atrybutu użytego w porównaniu. Postać warunku uniwersalnego jest następująca:

⟨*nazwa atrybutu*⟩ ⟨*operator*⟩ ⟨*wartość*⟩

przy czym operator należy do zbioru {=, <, <=, >, >=}.

Przykłady:

kolor_rany = pomarańczowy

poziom_leukocytów >9000

Warunek czasowy ma następującą składnię [4]:

⟨*kwantyfikator*⟩ ⟨*nazwa atrybutu*⟩|⟨*specyfikacja czasu*⟩|⟨*operator*⟩ ⟨*wartość*⟩

Kwantyfikatorem jest słowo kluczowe **EXISTS** lub **ALL**. Specyfikacja czasu wyznacza punkt lub przedział czasowy, w którym odbywa się sprawdzenie wartości danego atrybutu. Jest ona relatywna, tzn. podaje się ją w odniesieniu do tzw. stałej czasowej. Stałe czasowe oznaczają pewne istotne dla danego procesu chorobowego punkty czasu. Dla BW dotyczącej oparzeń, wybranej przy opracowywaniu prototypu systemu, przykładami stałych czasowych mogą być momenty: oparzenia, operacji, założenia cewnika itp. Bieżący moment czasu jest reprezentowany w systemie za pomocą słowa kluczowego **NOW**. Nadanie konkretnej wartości **NOW** odbywa się na początku każdej sesji konsultacyjnej, przy czym użytkownik może zaakceptować czas pobrany z systemu lub podać inną wartość. Taka elastyczność pozwala, na przykład, na prognozowanie przebiegu choroby z określonym wyprzedzeniem w czasie.

Przykłady warunków czasowych:

EXISTS *temperatura* [**TIME**⟩ *moment_oparzenia &*
TIME⟨= *moment_oparzenia*+3*D*] = *wysoka*
ALL *temperatura* [**TIME**⟩ *moment_oparzenia &*
TIME⟨= *moment_oparzenia*+3*D*] = *wysoka*

Pierwszy przykład dotyczy sprawdzenia, czy w ciągu pierwszych trzech dni po oparzeniu u pacjenta wystąpiła wysoka temperatura (nie wyklucza to wystąpienia w tym okresie innych wartości temperatury). W drugim przykładzie warunek jest spełniony, jeśli w danym czasie temperatura pacjenta była wyłącznie wysoka.

A oto przykład pełnej reguły:

IF *zakażenie_rany = tak* **AND**
(bolesność_rany = tak **OR** *kolor_rany = zielonkawy)* **AND**
ALL *temperatura* [**TIME**⟩*moment_oparzenia*] = *wysoka*
THEN
zakażenie_streptococcus_pyogenes 0.9;

Liczba 0,9 (zwana współczynnikiem pewności) określa stopień przekonania eksperta co do prawdziwości akcji danej reguły przy założeniu, iż jej przesłanka jest prawdziwa. Konieczność oceny stopnia wiarygodności reguł przez eksperta jest związana z obecnością w SYS8688 modelu probabilistycznego, służącego do oceny niepewności wiedzy [5].

SYSTEM BAZY DANYCH

Funkcje związane z gromadzeniem i uaktualnianiem danych o pacjentach realizuje system zarządzania bazą danych (SZBD). Struktura fizyczna zbiorów BD jest zgodna ze standardem dBasse III, jednak SZBD nie korzysta z tego oprogramowania ze względu na wymaganą szybkość działania systemu. Zastosowanie takiego standardu umożliwia komunikację SYS8688 z innymi pakietami przez system plików.

Atrybuty w BD są zorganizowane w kompleksy odpowiadające grupom powiązanych ze sobą parametrów klinicznych lub laboratoryjnych (np. kompleks parametrów badania morfologicznego). Kompleks może być niezależny lub zależny od czasu badania. W ostatnim przypadku możemy mieć do czynienia z dokładnością dobową lub godzinową gromadzenia danych, przy czym punkt pomiaru czasu może być identyczny dla całego zestawu parametrów i przechowywany jednokrotnie.

Dane są przechowywane w postaci trójek ⟨*O,S,T*⟩, czwórek ⟨*O,A,W,WP*⟩ oraz piątek ⟨*O,A,W,WP,T*⟩ lub ⟨*O,K,WP,T,Ż*⟩, gdzie:
O – obiekt (np. pacjent),
A – atrybut (np. parametr kliniczny),
W – wartość atrybutu,
WP – współczynnik pewności,
T – punkt czasowy,
S – stała czasowa,
K – konkluzja,
Ż – źródło informacji (lekarz lub SE).

System bazy danych SYS8688/BD pozwala uzyskiwać przekroje danych nie tylko w konwencjonalnym układzie (obiekt, atrybut), lecz także w układzie:

⟨*obiekt, atrybut, logiczne_określenie_punktu_czasu*⟩

gdzie logiczne_określenie_punktu_czasu jest zgodne ze składnią warunku czasowego przesłanki reguły [3].

Aby umożliwić opis poszukiwanych podzbiorów BD, zaprojektowano język kwerend do BD, którego składnię oparto na składni języka reguł. Kwerenda składa się z dwóch członów:
– pierwszy **FOR** określa warunek wyboru pacjentów (odpowiednik części **IF** reguły),
– drugi **WHAT** wskazuje atrybuty lub konkluzje, których wartości mają być przedstawione.
Rozszerzeniem w stosunku do składni reguł jest możliwość umieszczenia konkluzji w części **FOR** kwerendy.

Użytkownik wskazuje interesujący go podzbiór informacji definiując odpowiednią kwerendę. Jeśli uzyskany podzbiór z określonych powodów nie satysfakcjonuje użytkownika, to jest możliwe jego rozszerzenie lub zawężenie przez zmianę definicji kwerendy. Analiza warunku **FOR** przebiega analogicznie jak ewaluacja wyrażenia logicznego np. w języku C. Odpowiedzią na kwerendę jest liczebność uzyskanego zbioru i poszukiwana informacja. Można np. postawić pytanie:

FOR (* *dla pacjentów, u których* *)
wiek > 40 **AND**
płeć = kobieta **AND**
(zakażenie_rany = tak **OR**
EXISTS *temperatura* [**TIME**⟩ *moment_operacji*+5*D*] = *wysoka)*
WHAT (* *wyszukać następującą informację* *)
nazwisko,
ALL *poziom_potasu* [**TIME**⟩ *moment_operacji*+5*D*],
przebyte_choroby

Reasumując, dane zgromadzone w BD mogą być:
- źródłem informacji o indywidualnych obiektach (np. dla systemu ekspertowego podczas konsultacji), źródłem do generowania zestawień i statystyk medycznych, bazą dla przeprowadzania badań statystycznych (przy pomocy modułu statystycznego),
- podstawą do formułowania i weryfikacji stawianych przez ekspertów hipotez dotyczących reguł wnioskowania,
- bazą dla automatycznego generowania reguł z materiału, bez potrzeby ich zewnętrznego przekodowywania,
- podstawą do okresowej weryfikacji używanych reguł wnioskowania.

* * *

Pozostałe elementy systemu SYS8688, tj. system ekspertowy i system pozyskiwania wiedzy, a także komunikacja z użytkownikiem, zostaną omówione w drugiej części artykułu.

LITERATURA

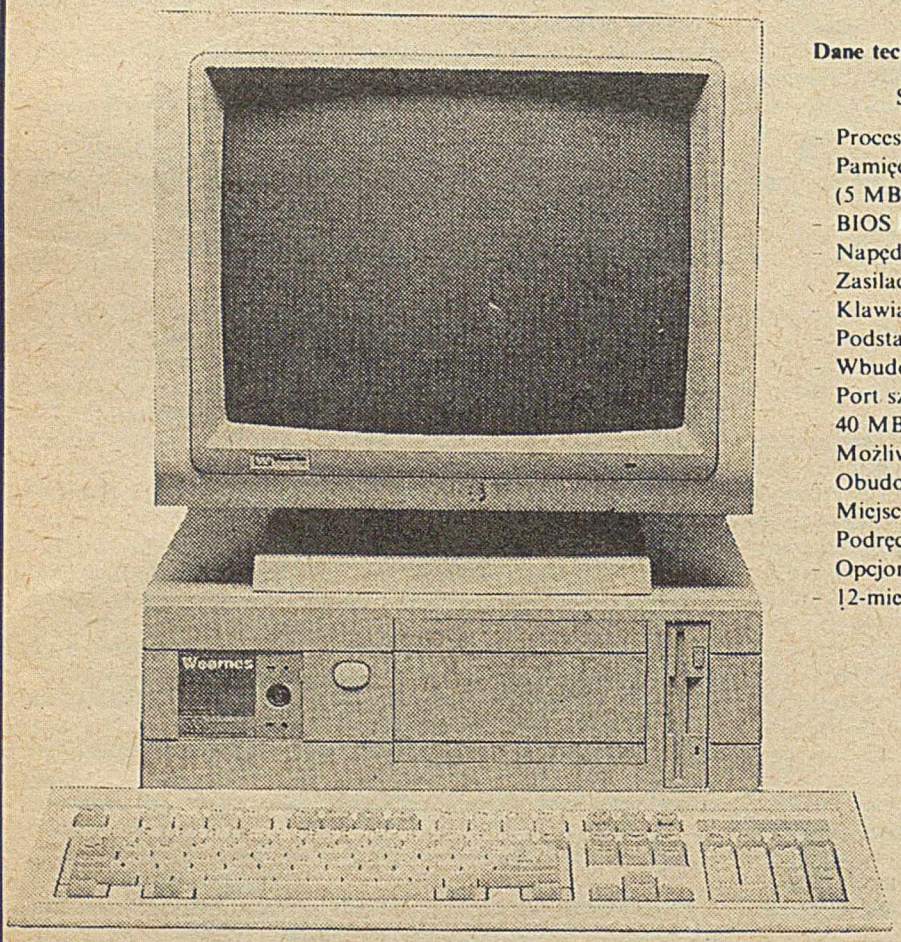
[1] Alty J. L., Coombs M. J.: Expert Systems. Concepts and Examples. National Computer Centre, Manchester, 1984
[2] Buchanan B. G., Shortliffe E. H.: Rulc-Based Expert Systems. Addison-Wesley, Reading (MA), 1984
[3] Dean T. L., McDermott D. V.: Temporall Data Base Management. Artificial Intelligence, Vol. 32, pp. 1–55, 1987
[4] Kłopotek M., Pacan A., Syropiatko E.: Uwzględnienie relacji czasowych wydarzeń w ekspertowo-statystycznym systemie diagnostyki medycznej SYS8688. Konferencja nt. Systemy eksperckie we wspomaganiu podejmowania decyzji. Poznań, 26–27 listopada 1987
[5] Shortliffe E. H.: Computer-Based Medical Consultations – MYCIN. American Elsevier, New York, 1987.

KOMPUTER
Z WBUDOWANĄ PRZYSZŁOŚCIĄ

Komputer zaprojektowany przez czołowego światowego producenta mikrokomputerów – firmę ADVANCED LOGIC RESEARCH i wyprodukowany przez firmę WEARNES TECHNOLOGY.

Komputer, którego możliwości i cena oszałamiają konkurencję.

Komputer, który nie zestarzeje się, dzięki możliwości rozbudowy do architektury 386SX i 486.



Dane techniczne:

SERIA WEARNES BOLDLINE „M"

- Procesor 80286 – 12.5 Mhz
- Pamięć 1 MB RAM, możliwość rozbudowy do 16 MB (5 MB na płycie głównej)
- BIOS Phoenix
- Napęd dyskietek 3,5" 1,44 MB
- Zasilacz 110 Watt
- Klawiatura 101 klawiszy
- Podstawka dla koprocesora matematycznego 80287
- Wbudowany sterownik dyskowy z przeplotem 1:1
- Port szeregowy i równoległy
- 40 MB dysk sztywny
- Możliwość korzystania z EMS 4.0
- Obudowa typu „compact"
- Miejsce na dwa napędy 5,25" o wysokości 1/2
- Podręcznik i dyskietka z programem konfiguracyjnym
- Opcjonalna rozbudowa do 386SX i 486
- 12-miesięczna gwarancja

Komputer z wbudowaną przyszłością!

BOLDLINE COMPUTERS – GRUPA MICOMP-TECH
Biuro Informacji Techniczno-Handlowej
ul. Astrów 7, 40–045 Katowice
tel. 51-86-28 (telefaks), teleks 315687 COMP PL

EO/1123/89

LECH BANACHOWSKI, JAROSŁAW DEMINET
MAREK J. LAO
Instytut Informatyki
Uniwersytet Warszawski

Propozycja nowego programu uniwersyteckich studiów informatycznych

W ostatnich latach coraz wyraźniej odczuwa się potrzebę zmiany obowiązującego programu studiów informatycznych, związaną z przeładowaniem obecnego programu, niezgodnością z obecnym stanem wiedzy oraz niedopasowaniem programu zajęć matematycznych do potrzeb informatyków. W niniejszym opracowaniu przedstawiamy propozycję nowego programu studiów, wzorowanego w dużej mierze na rozwiązaniach zalecanych porzez duże organizacje informatyczne (ACM, IEEE, Unesco-IFIP). W naszej propozycji wyróżniamy dwa bloki przedmiotów:

- ***blok podstawowy***, **którego celem jest wykształcenie świadomych programistów, zdolnych do kontynuowania bardziej specjalistycznych studiów; w bloku podstawowym zawiera się podstawowa nauka języków i technik programowania, przedmioty matematyczne oraz propedeutyczne wersje większości specjalistycznych przedmiotów informatycznych; blok podstawowy ma być ukierunkowany na praktykę informatyki;**
- ***blok zaawansowany***, **który ma dostarczyć studentom wiedzę niezbędną do uzyskania jednej z dwóch specjalności – programisty systemowego i informatyka-teoretyka; zamiast wydzielania konkretnych specjalności wprowadziliśmy studia zindywidualizowane pod opieką pracownika naukowo-dydaktycznego; w bloku tym studenci powinni poznać dogłębnie tylko kilka zagadnień, ale ich wiedza w tym zakresie powinna być poparta umiejętnościami praktycznymi (wprowadziliśmy więc pracownie programowania powiązane z poszczególnymi przedmiotami).**

Przedstawiona propozycja jest wstępnym i niepełnym programem studiów. Jest jednak bardziej elastyczna od obowiązującego programu studiów i umożliwia łatwiejsze wprowadzanie zmian w przyszłości. Potrzebne są jednak dalsze prace specjalistów ze wszystkich dziedzin informatycznych.

Materiał ten przedstawiamy do szerszej dyskusji dotyczącej nowego programu studiów. Autorzy nie pretendują do miana specjalistów ze wszystkich dziedzin informatycznych ani też dydaktyków informatyki. Wydaje się jednak, że materiał taki jak ten musiał powstać, by sprowokować poważniejszą dyskusję na temat zmian obowiązującego programu studiów.

NIEODZOWNOŚĆ ZMIANY PROGRAMU STUDIÓW

Obecnie obowiązujący program studiów powoduje duże niezadowolenie zarówno wśroód pracowników naukowo-dydaktycznych, jak i studentów. Oto główne zarzuty.

- **Program jest przeładowany** – przez pierwsze cztery lata studiów studenci mają po około 30 godzin zajęć tygodniowo. W efekcie pozostaje im bardzo mało czasu na pracę samodzielną, uczenie się materiału z wykładów, odrabianie zadań, opracowywanie i uruchamianie programów, czytanie literatury, własną refleksję i własne próby badawcze.
- **Obowiązujący obecnie samodzielny system pracowni programowania nie zdał egzaminu** – pracownie nie mają charakteru laboratoryjnego, lecz funkcjonują jako ćwiczenia audytoryjne uzupełniane specjalnymi wykładami, a to, co powinno być w pracowni najważniejsze (opracowywanie, uruchamianie, testowanie programów, dokumentowanie, eksperymentowanie z powszechnie stosowanymi programami) studenci wykonują samodzielnie w czasie własnym, często podczas wakacji. Stwarza to pewną fikcję i niejednolitość w realizacji pracowni, gdyż z jednej strony czas przeznaczony na nie trzeba – często czymś przypadkowym – wypełnić, z drugiej zaś – wiele przedmiotów (np. budowa translatorów, systemy operacyjne) do pełnego przyswojenia materiału wymaga uruchomienia choćby małych fragmentów systemów.
- **Treści matematyczne programu studiów i ich realizacja nie uwzględniają specyfiki studiów informatycznych.** Nacisk powinien być położony na stronę algorytmiczną wiedzy matematycznej i na środki matematyczne do opisywania zjawisk informatycznych. Obecny program powiela treści, których uczy się osoby specjalizujące się w matematyce z jednoczesnym zmniejszeniem wymagań. W tych okolicznościach studenci nie czują motywacji do uczenia się przedmiotów matematycznych.

SYLWETKA ABSOLWENTA

Unesco-IFIP wymienia wiele specjalności informatycznych, między innymi takie, jak programiści, koderzy, specjaliści od systemów informacyjnych. Według klasyfikacji przyjętej przez tę organizację uniwersyteckie kierunki informatyczne powinny kształcić następujących specjalistów:
– programistów systemowych (ang. system *programmers*) i inżynierów oprogramowania (ang. software *engineers*), którzy powinni mieć specjalistyczną wiedzę i umiejętności w zakresie potrzebnym do tworzenia efektywnego oprogramowania dotyczącego systemów operacyjnych, kompilatorów, interpreterów, oprogramowania transmisji danych itp.,
– informatyków-teoretyków (ang. *scientists, researchers)*, którzy powinni zajmować się dalszym rozwojem zasad i technik obliczalności czy informatyki (do tej grupy należy też chyba zaliczyć numeryków).

Programiści systemowi oraz informatycy-teoretycy to docelowe specjalności studiów. One właśnie określają sylwetkę absolwenta uniwersyteckich kierunków informatycznych. W trakcie studiów wszyscy studenci powinni uzyskać wiadomości i umiejętności, które pozwalałyby ich traktować jako świadomych programistów. Według klasyfikacji Unesco, programista umie przeanalizować zadanie dotyczące przetwarzania danych, zaprojektować lub wybrać odpowiedni algorytm, a także utworzyć program komputerowy o dobrej strukturze.

Ta dwustopniowa charakterystyka absolwenta uniwersyteckich studiów informatycznych (programista, a następnie programista systemowy lub teoretyk) powinna znaleźć swoje odbicie w przyjętym programie studiów. W proponowanym programie studiów polega to na wydzieleniu dwóch bloków: bloku podstawowego oraz bloku zaawansowanego.

Wykształcenie programistów jest głównym celem bloku podstawowego, ale nie celem końcowym. Dlatego blok podstawowy przygotowuje studentów do kontynuowania bardziej specjalistycznych studiów. Znajdują się tu również przedmioty matematyczne oraz propedeutyczne wersje większości specjalistycznych przedmiotów informatycznych. W bloku podstawowym jest zawarta podstawowa nauka języków i technik programowania. Blok podstawowy kończy się rocznym projektem **(projekt połówkowy)**.

Blok zaawansowany ma dostarczyć studentom wiedzę niezbędną do uzyskania jednej z dwóch specjalności: programisty systemowego lub informatyka-teoretyka (do tej grupy zaliczamy również numeryków). Zamiast wydzielania tych specjalności wprowadziliśmy studia zindywidualizowane pod opieką merytoryczną pracowników naukowo-dydaktycznych. Wydaje się bowiem, że w tak młodej dziedzinie jak informatyka potrzebni są absolwenci zróżnicowani, o zainteresowaniach zarówno teoretycznych, jak i praktycznych. Poza tym nawet w zakresie programowania systemów studenci powinni dogłębnie poznać kilka dziedzin, a nie wszystkie płytko. Blok zaawansowany składa się więc z wielu przedmiotów, spośród których studenci wybierają tylko niektóre. Przedmiotów tych nie należy jednak mylić z wykładami monograficznymi występującymi w obecnie obowiązującym programie studiów. Wykłady bloku zaawansowanego powinny być oferowane co roku (w niektórych przypadkach co dwa lata). Przypominają one bardziej istniejące kiedyś wykłady fakultatywne. Blok zaawansowany powinien się kończyć napisaniem pracy magisterskiej. Należy jedynie zwrócić uwagę na to, że opiekun naukowy nie musi być promotorem pracy

i może być przydzielany przez administrację. Jego zadania omówimy dalej.

BLOK PODSTAWOWY

Studenci po zakończeniu bloku podstawowego powinni:
- umieć pisać w rozsądnym czasie programy poprawnie działające i udokumentowane, a także umieć uzasadniać ich poprawność,
- umieć określać, czy napisali rozsądnie sprawny i dobrze zbudowany program, a także umieć modyfikować cudzy dobrze udokumentowany program,
- znać ogólne klasy zadań nadających się do rozwiązywania za pomocą komputerów i znać różne narzędzia do rozwiązywania tych zadań,
- rozumieć podstawy architektury komputerów,
- być przygotowani do podjęcia ukierunkowanych studiów, posiadać pewne podstawy matematyczne i informatyczne pozwalające na pogłębione studia.

Proponowany program bloku podstawowego powstał przez ustalenie tych działów i tematów, które powinny umożliwić wykształcenie programisty przygotowanego do świadomego podjęcia dalszych, bardziej ukierunkowanych studiów. Następnie tematy te zostały zgrupowane w przedmioty. Ustalono ich wzajemne zależności czasowe i w ten sposób uzyskano siatkę zajęć.

Materiał wykładany w bloku podstawowym nie powinien zawierać treści ulegających dużym zmianom w czasie. Powinien to być z natury materiał propedeutyczny, ukazujący pewne wzajemne powiązania całości informatyki. Powinna tu być zawarta cała część dotycząca stylu i technik programowania, a także niezbędne wiadomości matematyczne. W przedstawionym wykazie przedmiotów bloku podstawowego przy każdym przedmiocie podano w nawiasach liczbę godzin wykładów oraz ćwiczeń (łącznie z pracowniami).

INF1. Wstęp do informatyki (30+0)
INF2. Wstęp do programowania (30+45)
INF3. Struktury danych (30+45)
INF4. Techniki programowania I (30+45)
INF5. Techniki programowania II (30+45)
INF6. Wstęp do systemów komputerowych i operacyjnych (45+60)
INF7. Języki programowania i kompilatory (30+30)
INF8. Analiza algorytmów (30+30)
INF9. Bazy danych (30+30)
INF10. Metody optymalizacji (30+30)
INF11. Teoria obliczeń i Języków formalnych (30+30)
INF12. Projekt programistyczny (0+45)
INF13. Podstawy elektroniki cyfrowej (30+30)
MAT1. Matematyka dyskretna (60+60)
MAT2. Analiza matematyczna I (45+45)
MAT3. Analiza matematyczna II (45+45)
MAT4. Rachunek prawdopodobieństwa i statystyka (45+45)
MAT5. Numeryczna algebra liniowa (60+60)
MAT6. Metody numeryczne (60+60)

W tabeli przedstawiono przykładowy przydział przedmiotów do poszczególnych semestrów, po uwzględnieniu zależności czasowych przedmiotów bloku podstawowego oraz przy założeniu, że studia bloku podstawowego nie powinny trwać dłużej, niż pięć semestrów (połowa studiów).

BLOK ZAAWANSOWANY

Blok zaawansowany powinien umożliwić studentowi dogłębne poznanie kilku dziedzin informatyki praktycznej i teoretycznej. Jego zakończeniem powinna być praca magisterska.

Informatyka rozbudowuje się dość szybko. Nie jest już możliwe poznanie wszystkiego dokładnie. Dlatego też jest wskazany świadomy wybór kilku interesujących dziedzin i ich dokładne poznanie. Ważne jest, by student mógł dokonać wyboru spośród wielu interesujących dziedzin. By zapewnić rozwój bez zbyt wąskiej specjalizacji, wybór powinien być dokonany pod opieką merytoryczną pracownika. W proponowanym programie blok zaawansowany polega ogólnie na studiach indywidualnych pod taką opieką.

Wśród przedmiotów bloku zaawansowanego warto wyróżnić cztery kategorie:
- **przedmioty katalogowe,** czyli specjalistyczne wersje wykładów z dziedzin wstępnie sygnalizowanych w bloku podstawowym (np. budowa translatorów, teoria złożoności lub analiza numeryczna); przedmioty te powinny być w zasadzie oferowane w sposób ciągły (co roku lub nawet w każdym semestrze),
- **pracownie programowania** niektórych przedmiotów katalogowych,
- **wykłady monograficzne,** zmienne w czasie, wykłady specjalistów spoza uczelni itp.,
- **seminaria.**

W indywidualnym programie studiów powinny być reprezentowane przedmioty każdej z tych kategorii. Wydaje się też, że student powinien móc wybrać przedmiot katalogowy z przewidzianą pracownią, również w wersji bez pracowni. Należy zwrócić uwagę, że niektóre pracownie mogą mieć sens dopiero po zakończeniu (zaliczeniu) całego wykładu, inne w połowie wykładu dwusemestralnego, a jeszcze inne równolegle z wykładem.

Rozsądne jest ustalenie pewnych kryteriów doboru przedmiotów do studiów indywidualnych. Przyjmując jako zasadę, że wszystkie przedmioty bloku zaawansowanego są semestralne (choć być może w zwiększonym wymiarze godzin), można uznać, że do zaliczenia bloku zaawansowanego student powinien:
- w każdym semestrze uczestniczyć w ok. 15 godzinach zajęć tygodniowo,
- w każdym semestrze IV i V roku uczestniczyć w jednym seminarium i jednym wykładzie monograficznym (rocznym lub semestralnym); wydaje się, że za wykłady monograficzne można również uznać wszystkie wykłady katalogowe,
- w ciągu dwóch i pół roku bloku zaawansowanego zaliczyć co najmniej dwie roczne pracownie programowania,
- napisać pracę magisterską.

Innym wariantem studiów indywidualnych jest pozostawienie całkowitej swobody określenia tych studiów opiekunowi naukowemu studenta.

Przykładowy podział przedmiotów bloku podstawowego do poszczególnych semestrów w tygodniowym obciążeniu godzinowym (E oznacza egzamin)

Przedmiot	Semestr					
	I	II	III	IV	I	VI
INF 1	2+0 E					
INF 2	2+3 E					
INF 3		2+3 E				
INF 4			2+3 E			
INF 5				2+3 E		
INF 6				3+4 E		
INF 7					2+2 E	
INF 8					2+2 E	
INF 9					2+2 E	
INF 10					2+2 E	
INF 11					2+2 E	
INF 12					0+1	0+2
INF 13						2+2
MAT 1	2+2	2+2 E				
MAT 2	3+3 E					
MAT 3		3+3 E				
MAT 4			3+3 E			
MAT 5	2+2	2+2 E				
MAT 6			2+2	2+2 E		
Razem	11+10 3 egz.	9+10 4 egz.	7+8 2 egz.	7+9 3 egz.	10+11 5 egz.	2+4

OPIEKUN NAUKOWY STUDENTA

W przedstawianej propozycji student ma przydzielanego opiekuna (lub sam go wybiera) już pod koniec drugiego lub na początku trzeciego roku studiów. Opiekun po rozmowach ze studentem powinien określić prowadzącego projekt programistyczny (zgodnie z zainteresowaniami studenta), a następnie obserwując zainteresowania i uzdolnienia studenta określić program studiów indywidualnych. Program indywidualny nie powinien być jednak zbyt jednostronny; opiekun powinien zadbać o to, by student poznawał również inne działy informatyki niż te, które go najbardziej interesują (np. typowy programista systemowy może uczęszczać na wykłady z analizy numerycznej).

Ważne jest, by opiekun nie zmuszał studenta do studiów w swojej specjalności, lecz by ustalił program, który student będzie realizował z przyjemnością i pożytkiem. Opiekun naukowy nie musi prowadzić projektu ani pracy magisterskiej swojego podopiecznego. Jego zadaniem jest jedynie zadbanie o to, by program studiów był sensowny.

NIEKTÓRE PRZEDMIOTY BLOKU ZAAWANSOWANEGO

Przedstawiony poniżej wykaz katalogowy przedmiotów bloku zaawansowanego oraz pracowni nie jest w żadnym przypadku zamknięty ani pełny. Powinien on ulegać zmianom zgodnie z możliwościami uczelni lub rozwojem wiedzy. Autorzy opracowania zdają sobie sprawę z jego braków. Wydaje się, że tam gdzie to jest możliwe, student nie specjalizujący się w danym temacie powinien mieć możliwość zaliczenia tylko jednego semestru wykładu katalogowego. Tak więc programy wykładów powinny być tak ustalane, by pierwszy semestr dawał pewną zamkniętą wiedzę.

INF14a. Systemy operacyjne I (30+30)
INF14b. Systemy operacyjne II (30+30)
INF14P. Systemy operacyjne – pracownia programowania (0+45)
INF15. Wydajność systemów operacyjnych (30+30)
INF16a. Metody kompilacji I (30+30)
INF16b. Metody kompilacji II (30+30)
INF16P. Metody kompilacji – pracownia programowania (0+45), po zaliczeniu INF16a
INF17. Semantyka języków programowania (30+30)
INF17P. Semantyka języków programowania – pracownia programowania (0+45), po zaliczeniu INF17
INF18. Logiki programów (30+30)
INF19. Teoria obliczeń (30+30)
INF20. Teoria złożoności (30+30)
INF21a. Analiza algorytmów I (30+30)
INF21b. Analiza algorytmów II (30+30)
INF21P. Analiza algorytmów – pracownia programowania (0+45)
INF22. Optymalizacja (30+30)
INF22P. Optymalizacja – pracownia programowania (0+45)
INF23a. Grafika komputerowa I (30+30)
INF23b. Grafika komputerowa II (30+30)
INF23P. Grafika komputerowa – pracownia programowania (0+45)
INF24. Bazy danych (30+30)
INF25a. Sztuczna inteligencja I (30+30)
INF25b. Sztuczna inteligencja II (30+30)
INF25P. Sztuczna inteligencja – pracownia programowania (0+45), po zaliczeniu INF25a, równolegle z INF25b lub po nim
INF26. Modelowanie i symulacja (30+30)
INF26P. Modelowanie i symulacja – pracownia programowania (0+45)
INF27. Systemy mikrokomputerowe i sieci lokalne (30+30)
INF27P. Systemy mikrokomputerowe i sieci lokalne – pracownia programowania (0+45)
INF28. Sieci komputerowe
INF29. Organizacja pracy zespołowej i pracy ośrodka obliczeniowego (30+30)
INF29P. Organizacja pracy zespołowej i pracy ośrodka obliczeniowego (0+45)
INF30. Kombinatoryka (30+30)
INF31. Teoria grafów (30+30)
INF32a. Analiza numeryczna I (30+30)
INF32b. Analiza numeryczna II (30+30)
INF32P. Analiza numeryczna – pracownia programowania (0+45)
INF33. Równania różniczkowe (30+30)
INF34. Teoria kolejek (30+30)
INF35. Zastosowania informatyki (30+0)

UWAGI TECHNICZNE DOTYCZĄCE REALIZACJI PRZEDMIOTÓW

Do zrealizowania wszystkich celów bloku podstawowego będzie potrzebny duży zestaw sprzętu informatycznego różnego rodzaju. Najbardziej popularne w tej chwili mikrokomputery są bardzo pożądane przy studiowaniu szczegółów architektury komputerów oraz przy eksperymentach w dziedzinie systemów operacyjnych (INF6). Mogą też być z powodzeniem wykorzystane do prowadzenia zajęć laboratoryjnych w ramach wykładów INF2 i INF3. Ponieważ część zajęć pracowni powinna odbywać się bezpośrednio przy komputerze, jest wskazane, by było kilka laboratoriów mikrokomputerowych.

Nie wszystkie tematy omawiane na wykładach mogą być prezentowane za pomocą sprzetu mikrokomputerowego. Dlatego też jest ważne zapewnienie dostępu do średnich i dużych komputerów. Potrzebne są odpowiednie urządzenia wejścia-wyjścia oraz urządzenia systemowe zapewniające możliwość szybkiego obrotu prac studenckich, pracy interakcyjnej i pracy w ramach pracowni. Duże systemy powinny być wyposażone w oprogramowanie służące jako ilustracja wykładów oraz w języki programowania przewidziane w programie.

Przedmiot INF13 z podstaw elektroniki cyfrowej wymaga odpowiedniego zaplecza technicznego. Laboratorium elektroniczne z kilkoma stanowiskami montowania układów z układów scalonych oraz układów o dużym stopniu scalenia pozwala na praktyczne sprawdzenie zdobytych wiadomości oraz na budowę bardzo małego komputera z części podstawowych. W odróżnieniu od dużego sprzętu informatycznego, koszt takiego laboratorium jest bardzo mały.

Efektywne nauczanie i badania w informatyce wymagają wyposażenie laboratoryjnego. Informatyka jest nauką w dużej mierze empiryczną, która obejmuje realizowanie procedur i studiowanie przebiegów procesów o znanych właściwościach teoretycznych. Dlatego ważne jest, by studenci i pracownicy mieli dostęp do sprzętu laboratoryjnego porównywalnego z tym, jaki jest potrzebny w fizyce, chemii lub naukach biologicznych.

Wstępne oszacowanie czasu dostępu do komputera (terminala, mikrokomputera) potrzebnego do wykonania zadań przewidzianych w proponowanym programie studiów, przypadającego na jednego studenta w ciągu tygodnia wynosi:

- I rok – ok. 5 godz. (w tym 2 godz. pracowni),
- II rok – ok. 8 godz. (w tym 4 godz. pracowni),
- III rok – blok podstawowy (projekt) – ok. 5 godz.,
- każdy rok bloku zaawansowanego – ok. 12 godz. (w tym 4 godz. pracowni),
- praca magisterska – ok. 8 godz.

Ponieważ wykłady są bardzo skondensowane, jest niezbędne przygotowanie tylu egzemplarzy podstawowych podręczników do wykładu, ilu studentów jednocześnie uczęszcza na ten wykład. Przy tym założeniu część materiału, traktowana jako uzupełniająca lub pogłębiająca, może być zadawana do samodzielnego przygotowania przez studentów.

W dużej części wykładów zwłaszcza informatycznych, przekazuje się pewne dane zbiorcze (schematy systemu komputerowego, dane statystyczne). Każdą salę wykładową należałoby więc zaopatrzyć w rzutnik, co znacznie ułatwia przekazywanie wiadomości. Ten sposób nadaje się też do prowadzenia wykładów matematycznych, na których większość definicji i twierdzeń (bez dowodów) może być przedstawiona na ekranie zamiast na tablicy. Studenci powinni sami dopisywać do nich komentarz. Wymagane są jednak dodatkowe materiały: studenci powinni dostać odbitki przezroczy przed wykładem, by nie tracili czasu na ich przepisywanie lub przerysowywanie i mogli skupić uwagę na przekazywanym komentarzu słownym. Oznacza to konieczność łatwego dostępu wykładowcy do urządzeń powielających. Integralną częścią przygotowań do wprowadzenia nowego programu powinno być napisanie i opublikowanie skryptów.

Sposób prowadzenia pracowni wymaga przygotowania instrukcji przeprowadzenia eksperymentu przed każdymi zajęciami. Powielone materiały określające niezbędną wiedzę (lub odwołania do literatury) wraz z poleceniami określającymi sposób realizacji zadania muszą być dostępne dla studentów przed zajęciami. Na zajęcia pracowni student powinien przyjść przygotowany do przeprowadzenia konkretnego zadania. Prowadzący zajęcia sprawdza przygotowanie studentów i ewentualnie koryguje ich błędy przed rozpoczęciem zajęć przy komputerze lub stanowisku elektronicznym. Taki sposób prowadzenia pracowni jest powszechny na uczelniach technicznych, kierunkach fizycznych, chemicznych i innych eksperymentalnych. Wydaje się, że powinien być stosowany również w informatyce.

W kolejnych częściach artykułu zostaną szczegółowo omówione przedmioty obu bloków oraz podane przykładowe zestawy przedmiotów bloku zaawansowanego, odpowiadające wybranym specjalizacjom.

Wychodząc naprzeciw zapotrzebowaniu środowiska redakcja nasza oferuje łamy INFORMATYKI na bezpłatne zamieszczanie anonsów o imprezach informatycznych organizowanych na terenie kraju.

Zwracamy się do organizatorów konferencji, wystaw, konkursów i sympozjów z prośbą o przysyłanie do redakcji krótkich (pół strony maszynopisu) informacji o planowanych imprezach. Teksty powinny zawierać datę i miejsce imprezy, krótką charakterystykę jej tematyki oraz adres organizatorów.

Oczekujemy na zgłoszenia.

AE AKERMANN ELECTRONIC SA

Twoim dostawcą najnowocześniejszej elektroniki

Nasza firma **AKERMAN ELECTRONIC** ma 35 lat doświadczeń w handlu wyrobami nowoczesnej elektroniki. Podstawowymi zasadami w jej działalności są niezawodność oraz wysoka jakość sprzedawanych wyrobów.

Nasza oferta, obejmująca najnowocześniejsze, a jednocześnie bardzo dojrzałe pod względem eksploatacyjnym rozwiązania techniczne, stanowi odpowiedź na zgłaszane przez klientów zapotrzebowanie. Podstawowym zadaniem w działalności firmy jest wszechstronne zaspokajanie potrzeb klientów.

Na rynku polskim mamy wyłączność sprzedaży wyrobów znanej firmy **MICROTEK INTERNATIONAL INC.** z Tajwanu. W Warszawie dysponujemy własnym serwisem, gwarantujemy pełen zakres obsługi technicznej sprzedanych wyrobów z maksymalnym uwzględnieniem indywidualnych wymagań klientów. Serwis ten jest dodatkowym świadczeniem firmy, umożliwiającym rozwiązywanie na miejscu wszystkich problemów natury technicznej, jakie wynikają w trakcie użytkowania dostarczonych wyrobów.

Firma **AE AKERMANN ELECTRONIC SA** jest przedsiębiorstwem działającym w dziedzinie szczytowych osiągnięć współczesnej elektroniki, którego celem jest nie tylko oferowanie wyrobów o najnowocześniejszych rozwiązaniach technicznych, ale również świadczenie usług związanych z użytkowaniem tych wyrobów.

Oferta firmy obejmuje dostawy szerokiego asortymentu półprzewodników, wyposażenia i oprzyrządowania systemów komputerowych a także komputerów osobistych. W tym ostatnim asortymencie firma połączyła najlepsze rozwiązania technologii sprzętu komputerowego i oprogramowania z licznymi dodatkowymi udogodnieniami, tworząc własną rodzinę komputerów osobistych AE. Rodzina ta obejmuje pełną skalę modeli uwzględniających potrzeby różnych kategorii użytkowników.

AE AKERMANN ELECTRONIC SA ma wyłączność sprzedaży wyrobów znanej firmy **MICROTEK INTERNATIONAL INC., R.O.C. Taiwan**

MICROTEK jest innowacyjnym przedsiębiorstwem z dziedziny najnowocześniejszej elektroniki, które specjalizuje się w opracowywaniu i produkcji wyrobów służących przetwarzaniu informacji.

Swą działalność w dziedzinie opracowania nowych wyrobów **MICROTEK** koncentruje na dwóch podstawowych obszarach: automatyzacji prac inżynierskich oraz automatyzacji prac biurowych.

Spośród wyrobów firmy **MICROTEK** należy w pierwszym rzędzie wymienić dwa, których rozwiązania zdecydowanie przewyższają osiągnięcia innych firm. Są to:

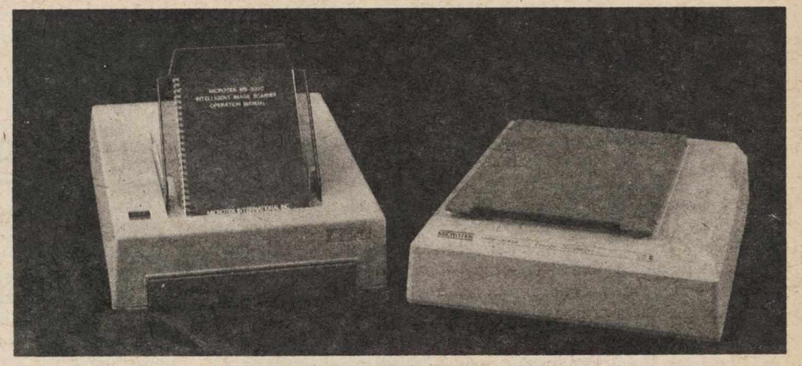
- **emulatory mikroprocesorów serii MICE** (Micro-In-Circuit Emulator), współpracujące ze standardowymi komputerami osobistymi typu PC, przeznaczone do projektowania i uruchamiania urządzeń wykorzystujących mikroprocesory,
- **skanery serii MSF-300** do komputerowego odczytu obrazów, obejmujące model dostosowany do wymagań poligrafii profesjonalnej oraz model uwzględniający wymagania komputerowego przygotowania publikacji.

MICROTEK MICE

Urządzenia MICE (Micro-In-Circuit Emulator) są układowymi emulatorami mikroprocesorów. Dołączone do badanego obwodu i współpracujące ze standardowymi komputerami osobistymi typu PC znakomicie upraszczają i usprawniają pracę projektanta urządzeń wykorzystujących mikroprocesory. Pojawienie się emulatorów serii **MICROTEK MICE** stanowiło rewolucję w metodach projektowania obwodów mikroprocesorowych. Są to bardzo zwarte i tanie urządzenia przenośne w formie przystawek do typowych komputerów PC. Praktycznie każdy z dostępnych na rynku i powszechnie stosowanych mikroprocesorów, takich jak INTEL, MOTOROLA, ZILOG, HITACHI, NATIONAL, itp., może być emulowany przez odpowiedni model z serii **MICE**. Wszechstronność ta sprawia, że **MICE** firmy **MICROTEK** jest narzędziem bardzo wydajnym i wygodnym w użytkowaniu, a także znacznie tańszym od dotychczas stosowanych rozwiązań.

Podstawowa charakterystyka **MIKROTEK MICE:**

- emulacja w czasie rzeczywistym,
- duża rozdzielczość obrazowania map pamięci z rozbudowanymi funkcjami zabezpieczającymi,
- śledzenie wykonywania programu (*trace*) w czasie rzeczywistym,
- rozbudowany system pułapek z możliwością odtwarzania zatrzymanego programu w przód i w tył,
- wieloprocesowa synchronizacja wejścia-wyjścia,
- rezydentny asembler liniowy, deasembler, zegar wykonania oraz wiele innych rozwiązań.



USD (Universal Symbolic Debugger – uniwersalny symboliczny program uruchomieniowy) jest w pełni zintegrowanym pakietem programowym, specjalnie zaprojektowanym w celu zwiększenia możliwości uruchamiania, jakimi dysponują emulatory serii **MIKROTEK MICE**. Pakiet USD jest dołączony do każdego emulatora **MICE**.

Skanery MICROTEK

Skanery firmy **MICROTEK** są szczytowym osiągnięciem komputerowego odczytu obrazów. Niezależnie od tego, do czego chciałbyś wykorzystać skaner, zainteresuj się ofertą naszej firmy!

Skanery **MICROTEK** umożliwiają komputerom osobistym odczytywanie i przetwarzanie wszelkiego rodzaju druków, rysunków, fotografii, rękopisów i innych dokumentów. Skanery te, eksploatowane za pomocą pakietów programowych EYESTAR oraz VERSASCAN PLUS, charakteryzują się takimi cechami jakościowego przetwarzania obrazów, jak dostosowanie jasności i kontrastu, skalowanie oraz uwzględnianie szerokiego zakresu półtonow. Istnieje rowniez możliwość uzyskania dodatkowego oprogramowania do obróbki rysunków technicznych i do automatycznego rozpoznawania znaków drukowanych.

Skaner MSF-300G spełnia wymagania profesjonalnej techniki poligraficznej. Pozwala na ciągłe odtwarzanie w fotografiach półtonów, rozpoznając do 256 odcieni szarości. Zalecamy użycie tego modelu do współpracy z najwyższej klasy urządzeniami do przetwarzania obrazów.

Skaner MSF 300A jest urządzeniem przystosowanym do współpracy z systemami do komputerowego przygotowania wydawnictw (desktop publishing). Zapewnia możliwość dysponowania większą skalą jasności, kontrastu i półtonów, niż spotkać to można w innych skanerach do skomputeryzowanego składu wydawnictw.

Rodzina komputerów osobistych AE

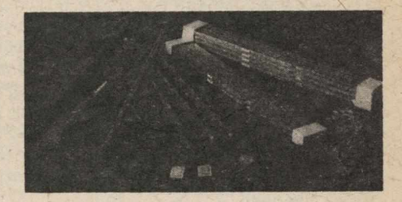
Komputery osobiste firmy AE zapewniają pełną realizację wszystkich rodzajów zastosowań. Stanowią one idealne narzędzie zwłaszcza dla takich zastosowań, jak systemy biurowe i przemysłowe oraz systemy operacji finansowych, a także mogą spełniać funkcje stacji roboczych w obszarze zastosowań CAD/CAM. W każdym przypadku poszukiwań sprzętu tej kategorii natychmiast zwróć się do nas z żądaniem szczegółowej oferty.

Przedstawimy Ci szeroki zakres dodatkowych możliwości tych komputerów. Rodzina komputerów osobistych AE zapewni Ci rozwiązanie najbardziej nowoczesne.

Półprzewodniki

Wszędzie tam, gdzie pojawia się zapotrzebowanie na materiały niezbędne do budowy systemów i urządzeń elektronicznych, pojawić się musi również firma **AE AKERMANN ELECTRONIC** jako Twoje niezawodne źródło zaopatrzenia. Nasz magazyn, zawierający zapas elementów wartości 3 mln dolarów, jest gwarancją szybkiej realizacji każdego zamówienia.

Niezależnie od tego, czy rozpoczynasz swą działalność produkcyjną od elementów najprostszych, czy też potrzebujesz elementów optoelektronicznych lub najnowocześniejszych mikroprocesorów, **firma AE AKERMANN ELECTRONIC** potrafi natychmiast zaspokoić Twoje potrzeby i pozostanie Twoim niezawodnym stałym dostawcą tej kategorii wyrobów. Jest oczywiste, że w naszej działalności ceny dostosowujemy zawsze do aktualnej sytuacji rynkowej.



Po szczegółowe informacje prosimy zwracać się do centrali firmy w Szwajcarii:

AE AKERMANN ELECTRONIC SA
Av. General Guisan 58
CH – 1800 Vevey/Switzerland
Telefon: 021/921–74–81
Telefaks: 021/921–83–30
Teleks: 451 122 ELVV CH

O/1/90

do naszego przedstawicielstwa w Polsce:

MITEL sp. z o.o.
02–594 Warszawa
ul. Bruna 10 m 29
Telefon: 25–70–38
Teleks: 815752

Mitel sp.z oo

JANUSZ ZALEWSKI
Instytut Energii Atomowej
Otwock-Świerk

Okres istnienia oprogramowania (2)

W drugiej części artykułu omówiono przykłady procesów produkcyjnych okresu istnienia oprogramowania. Każdy z trzech procesów ścisłego wytwarzania oprogramowania – formułowanie wymagań, projektowanie i implementowanie – ma kluczowe znaczenie dla jakości ostatecznego produktu. Wydaje się, że zasady implementacji, obejmujące takie czynności, jak wytworzenie kodu źródłowego (kodowanie), opracowanie danych testowych, wygenerowanie kodu wynikowego, tworzenie dokumentacji eksploatacyjnej, planowanie i wykonywanie integrowania modułów, są najlepiej poznane i najwszechstronniej stosowane. Poniżej zwrócę więc uwagę przede wszystkim na początkowe procesy produkcyjne: specyfikowanie wymagań i projektowanie.

PROCES SPECYFIKOWANIA WYMAGAŃ

W jednej z pierwszych norm [4] dotyczących wytwarzania oprogramowania, obejmującej specyfikowanie wymagań programowych, określono przede wszystkim konieczne cechy takiej specyfikacji:

- jednoznaczność (ang. *unambiguity*),
- zupełność (ang. *completeness*),
- sprawdzalność (ang. *verifiability*),
- niesprzeczność (ang. *consistency*),
- modyfikowalność (ang. *modifiability*),
- śladowalność (ang. *traceability*),
- użyteczność w fazach eksploatacji i konserwacji (ang. *usability*).

Specyfikowanie wymagań programowych według IEEE

Każda z tych cech jest opisana bardziej szczegółowo. Jednoznaczność specyfikacji najłatwiej osiągnąć stosując do specyfikowania język formalny. Tej dziedzinie poświęca się współcześnie niezmiernie wiele uwagi, jednak większość metod formalnych udaje się stosować jedynie do prostych przypadków por. [3], [7], [11], a stopień ich komplikacji na ogół wyklucza użycie w rzeczywistych przedsięwzięciach programistycznych. Do **zupełności** specyfikacji można zbliżyć się eliminując z niej maksymalnie tzw. TBD's (ang. *to be determined*), tj. fragmenty do późniejszego określenia. **Weryfikowalność** specyfikacji polega na tym, że muszą istnieć sposoby sprawdzenia, czy gotowy produkt spełnia poszczególne wymagania zawarte w specyfikacji. Przykładowo, za nieweryfikowalne należy uznać następujące stwierdzenia:
– produkt powinien mieć dobre sprzężenie użytkowe (nie wiadomo, co znaczy – dobre)
– program nie może wejść w nieskończoną pętlę (niemożliwe może być udowodnienie, że program nie zawiera pętli nieskończonych).

Niesprzeczność specyfikacji jest osiągnięta wtedy, gdy żadne z indywidualnych wymagań nie wchodzą ze sobą w kolizję, co może się zdarzyć, jeśli np.:
– w dwóch różnych miejscach specyfikacji żąda się uzyskania dwóch przeciwstawnych cech tego samego obiektu,
– narzuca się niezgodne wymagania na tę samą czynność (np. do wykonania w tym samym czasie),
– dwa lub więcej wymagań opisuje ten sam obiekt, czyniąc to różnymi językami.
Modyfikowalność specyfikacji jest znacznie ułatwiona, jeśli dokument specyfikacyjny ma dobrą organizację, tj. klarowny podział na rozdziały, spis treści, indeks, słownik itp., i nie zawiera nadmiarowości (redundancji). Choć sama nadmiarowość nie jest błędem, łatwo prowadzi do błędów. **Śladowalność** polega na tym, że pochodzenie i przyczyna (a więc ślad, stąd – śladowalność) każdego wymagania są na tyle jasno określone, że można zanalizować (prześledzić) sposób tworzenia danego wymagania i odtworzyć jego źródło. Dotyczy to zarówno śladów wstecz, jak i wprzód, ponieważ specyfikacja jest dokumentem wielokrotnie zmienianym przed skrystalizowaniem się ostatecznej wersji.

Podkreśla się, że dokument ten powinien być opracowany łącznie przez nabywcę (ang. *customer*) i dostawcę (ang. *supplier*) oprogramowania, a zawarte w nim wymagania powinny określać skutki działania oprogramowania, a nie sposoby ich osiągnięcia. Wyliczono też główne sposoby specyfikowania wymagań oraz – kilka rodzajów wymagań narzuconych na oprogramowanie, z których najważniejszymi są wymagania:
– funkcjonalne, określające, co oprogramowanie ma robić,
– wydajnościowe, np. szybkość, czas reakcji, liczba i rozmiar obsługiwanych plików, liczba jednoczesnych użytkowników itp.,
– sprzężeniowe, dotyczące zasad współpracy z użytkownikami, sprzętem, innym oprogramowaniem, systemem komunikacyjnym itd.
Opierając się na tych zasadach w normie [4] podano przykładowy kształt specyfikacji wymagań programowych oraz ich zawartości.

Specyfikowanie wymagań według EWICS

W bardzo blisko związanym z tą normą rozdziale dotyczącym specyfikacji, zawartym w zaleceniach [9], proces specyfikowania nieco rozszerzono. Podobnie, przez specyfikację rozumie się określenie wymagań, jakie powinno spełniać oprogramowanie zdaniem nabywcy, i podobnie zaleca się, aby specyfikacja zawierała jedynie wymagania, a nie proponowała rozwiązań, gdyż mogą one wprowadzać istotne ograniczenia. Podkreśla się jednak, co jest niesłychanie ważne, że specyfikacja wymagań powinna być jedynym środkiem porozumienia między nabywcą a dostawcą (lub wykonawcą) systemu. Choć uważa się, że specyfikacja rzadko jest dokumentem ustalonym raz na zawsze (jest to oczywiste, ponieważ będąc przedmiotem zlecenia, stanowi przedmiot ciągłych negocjacji między zleceniodawcą a zleceniobiorcą), powinna w pewnym momencie stać się dokumentem zamrożonym. Od tej chwili wszelkie zmiany specyfikacji winny być przedmiotem formalnej procedury zmian (ang. *change control procedure*).

Bardziej istotne jest jednak rozszerzenie rozumienia specyfikacji polegające na stwierdzeniu, że pełna specyfikacja powinna odnosić się nie tylko do tzw. systemu docelowego (ang. *target system*), będącego przedmiotem zamówienia czy dostawy, lecz też do warunków produkcji i warunków eksploatacji tego systemu. Z tego powodu, specyfikacja wymagań, oprócz wymagań dotyczących samego oprogramowania czy systemu docelowego, musi zawierać wymagania narzucane na środowisko eksploatacyjne, a także na środowisko produkcyjne, tzn. całe przedsięwzięcie (ang. *project*). Zasadnicza część odpowiada więc wymienionemu założeniu, określając, jak formułować wymagania dla systemu docelowego, jego środowiska, oraz – wymagania odnoszące się do całego cyklu produkcyjnego (przedsięwzięcia) i środowiska, w jakim jest realizowany.

Spośród zaleceń dotyczących systemu docelowego, najciekawsze są związane z wymaganiami decydującymi o rzetelności systemu (ponieważ tego dotyczy całość materiału [9]), a więc uwzględniającymi takie cechy, jak niezawodność, bezpieczeństwo i poufność, lecz także – adaptowalność, dyspozycyjność i konserwowalność systemu. Odnośnie pozostałych trzech aspektów tego modelu specyfikacji, zalecenia dotyczą tylko cech specyficznych, np. dla środowiska eksploatacyjnego szczególnie istotne jest bezpieczeństwo, a dla całego przedsięwzięcia (tzn. cyklu produkcyjnego) i jego środowiska – zagadnienie kontroli jakości. Oczywiście, główną część specyfikacji powinny stanowić tradycyjne wymagania dotyczące funkcji, wydajności i sprzężeń systemu docelowego – co też wyraźnie podkreślono w zaleceniach.

Normy a techniki i metody

Technologia tworzenia specyfikacji wymagań programowych (i systemowych) i automatyzacja tego procesu jest w stanie intensywnego rozwoju, dlatego też może się wydawać, że normy te są ubogie, nie

zawierają istotnych wskazówek i można je znacznie rozwinąć, np. wzbogacając o metody, języki (techniki) i narzędzia specyfikacyjne [8]. Należy jednak pamiętać, że wtedy nie byłyby już normami, lecz praktyką zalecaną przez zwolenników tych a nie innych metod, technik lub narzędzi. Pokusa znormalizowania jednej, wybranej metody lub techniki jest tym większa, że wytwórcy oprogramowania intensywnie dążą do posiadania narzędzi do automatycznego przekształcania specyfikacji formalnych na kod. Choć niejedną z tych technik stosuje się w produkcji oprogramowania już od wielu lat, żadne z dotychczas proponowanych podejść nie jest jeszcze na tyle dojrzałe, aby można je uznać za warte zalecenia w standardowej postaci. Stosunkowo najbliższa osiągnięcia tego stanu jest metoda VMD (ang. *Vienna Development Method*), której prace standaryzacyjne już rozpoczęto, lecz jej właśnie brakuje przede wszystkim narzędzi – a one głównie świadczą o dojrzałości metody.

PROCES PROJEKTOWANIA

Proces projektowania uważa się za decydujący dla powodzenia całego przedsięwzięcia (tj. wytworzenia produktu o wysokiej jakości), ponieważ polega on na przekształceniu specyfikacji wymagań, określającej **co należy zrobić**, na specyfikację projektową, określającą **jak należy to zrobić.** Na ogół istnieje zgodność co do podziału procesu projektowego na projektowanie ogólne, tj. projektowanie architektury oprogramowania, a więc funkcji i struktury jego składowych, oraz – projektowanie szczegółowe, tj. struktur danych i algorytmów tworzących poszczególne składowe.

Opis projektu według IEEE

Do tych dwóch podstawowych czynności procesu projektowania w normie [6] dodano jeszcze następujące:
- analiza przepływu informacji,
- projektowanie sprzężeń,
- projektowanie bazy danych,
- wybór lub opracowanie algorytmów.

Wynikiem projektowania szczegółowego, opartego m.in. na specyfikacji wymagań, opisie architektury, opisie bazy danych, opisie sprzężeń i opisach algorytmów, powinien być niezwykle ważny dokument – **opis projektu oprogramowania** (ang. *software design description*). Według definicji [5] opis projektu oprogramowania jest to reprezentacja oprogramowania utworzona w celu ułatwienia analizy, planowania, implementowania i podejmowania decyzji. Opis projektu służy jako środek przekazywania informacji o projekcie oprogramowania.

Zasady i sposoby opisu projektów są objęte samodzielną normą [5], którą naprawdę warto przestudiować dokładniej. Przede wszystkim należy ponownie podkreślić, że ta norma także nie wyróżnia żadnej szczególnej metodyki ani techniki projektowania, pozostawiając ich wybór projektantom. Podaje natomiast zalecenia dotyczące zawartości i organizacji dokumentu zwanego opisem projektu oprogramowania. Elementy składowe projektu nazywa się w niej **jednostkami projektowymi** (ang. *design entities*). Przykładami jednostek projektowych mogą być systemy, podsystemy, zbiory danych, moduły, programy, procesy, podprogramy i procedury. Jednostki projektowe opisuje się za pomocą atrybutów, tj. określonych charakterystyk lub właściwości. W normie zaproponowano opis jednostek projektowych za pomocą dziesięciu atrybutów spełniających następujące kryteria:
- uwzględnienie atrybutu jest konieczne we wszystkich przedsięwzięciach programistycznych,
- niepoprawna specyfikacja wartości atrybutu mogłaby spowodować defekt opracowywanego oprogramowania,
- atrybut opisuje informację ściśle projektową, a nie informację tylko związaną z projektem.

Tak wybrana grupa dziesięciu atrybutów obejmuje następujące:
1) identyfiktor, tj. jednoznaczna nazwa charakteryzująca jednostkę,
2) typ, tj. określenie rodzaju jednostki,
3) cel (tzn. odpowiedź na pytanie o sens istnienia jednostki),
4) funkcja, tj. stwierdzenie dotyczące czynności wykonywanych przez jednostkę, a dokładniej transformacji danych wejściowych, aby uzyskać pożądane dane wyjściowe,
5) jednostki składowe, tworzące strukturę rozważanej jednostki,
6) zależności, tj. hierarchiczne związki z innymi jednostkami,
7) zasoby, tzn. zewnętrzne wobec projektu źródła zasilające, niezbędne usługi itp. (np. drukarki, pamięci, biblioteki, procesory itd.),
8) sprzężenia, tj. opis zasad współpracy z innymi jednostkami,
9) przetwarzanie (tzn. opis metody prowadzącej do wykonania funkcji),
10) dane (wewnętrzne jednostki).

Zalecono szczegółowy opis atrybutów, który np. dla sprzężenia powinien zawierać m. in. mechanizmy wywoływania jednostki, wykonywania przerwań, przekazywania parametrów i komunikatów, korzystania ze wspólnych danych, bezpośredniego dostępu do danych, składnię komunikatów, formaty danych, zakresy wartości wejściowych i ich znaczenie, kody błędów, format obrazu, język komunikacji itd.

Powyższa informacja w postaci atrybutów może być powiązana w opisie projektu różnymi sposobami. Proponowane zasady polegają na wyborze niektórych wymienionych cech i połączeniu ich w opis wyższego poziomu, np.:
- opis dekompozycyjny, uwzględniający identyfikator, typ, cel, funkcję i składowe każdego modułu,
- opis współoddziaływań, określający dla wszystkich modułów ich identyfikatory, typy, cele, zależności i zasoby,
- opis zewnętrzny, obejmujący identyfikator, typ i sprzężenia każdego modułu,
- opis wewnętrzny, na który składają się identyfikatory, przetwarzanie i dane wszystkich modułów.

Zalecenia projektowe EWICS

Nieco inaczej do procesu projektowania podchodzi się w odpowiednim rozdziale zaleceń [10]. Nie należy zresztą temu się dziwić, ponieważ dotyczą one przede wszystkim projektowania systemów z uwzględnieniem bezpieczeństwa. Podstawowa różnica polega tu na zwróceniu większej uwagi nie na sam proces projektowania, lecz na czynności przygotowawcze. Sposób postępowania określono w pięciu zasadach projektowych:
1) niezależna analiza bezpieczeństwa, polegająca na sporządzeniu i zbadaniu listy możliwych wypadków, zagrożeń itp., i powiązaniu ich możliwych przyczyn z oprogramowaniem,
2) rozdzielenie perojektu oprogramowania na część krytyczną dla bezpieczeństwa i niekrytyczną,
3) zapewnienie maksymalnie wysokiej niezawodności części krytycznej,
4) zapewnienie bezpieczeństwa kosztem niezawodności lub innych atrybutów jakości,
5) ciągłe nadzorowanie części krytycznej oprogramowania.

Realizację tych zasad proponuje się w czterech krokach, z których pierwsze dwa nie mają charakteru ściśle projektowego, lecz przygotowawczy:
- wszechstronna analiza bezpieczeństwa komputeryzowanej instalacji i jej otoczenia (wynikiem tego kroku jest wyróżnienie celów bezpieczeństwa, rozumianych jako podział wszystkich możliwych stanów na bezpieczne i niebezpieczne, oraz sformułowanie ograniczeń projektowych),
- analiza specyfikacji projektowej pod względem bezpieczeństwa (wynikiem tego kroku jest lista wymagań projektowych, zwiększających bezpieczeństwo).

Krok trzeci, ściśle projektowy, polega na opracowaniu dokumentu będącego projektem systemu docelowego, na podstawie: specyfikacji wymagań funkcjonalnych, sprzężeniowych i in., oraz ograniczeń i wskazówek sformułowanych w dwóch poprzednich krokach. W tym kroku można oczywiście wprost stosować sposób przygotowywania projektu zalecany w normie [5]. Ostatni krok polega na zweryfikowaniu dokumentów projektowych pod względem zgodności ze specyfikacją oraz z wymaganiami bezpieczeństwa określonymi w dwóch krokach przygotowawczych. Po wprowadzeniu poprawek i ponownym zweryfikowaniu, projektowanie uważa się za zakończone.

Aby ułatwić realizację projektów, opracowano szczegółowy poradnik zalecanych technik projektowania [1], grupując je w pięciu rozdziałach poświęconych kolejno: analizie bezpieczeństwa, unikaniu defektów, wykrywaniu defektów, wykrywaniu uszkodzeń i neutralizowaniu uszkodzeń (ang. *failure containment*). Każdą technikę scharakteryzowano dość szczegółowo, podając najbardziej istotne informacje w następującej kolejności: cel, opis, warunki użycia, główne wady i podstawowe zalety, powiązanie z innymi metodami, dostępne narzędzia, źródła literaturowe.

W rozdziale dotyczącym analizy bezpieczeństwa umieszczono te metody, które zaleca się stosować w krokach przygotowawczych

projektowania, do zidentyfikowania zagrożeń i ich źródeł. Do takich metod należą m.in.: diagramy przyczyn i skutków (ang. *cause-consequence diagrams*), drzewa zdarzeń i drzewa defektów (ang. *event-tree analysis, fault-tree analysis*), karty gry w Go (ang. *Go charts*) i in. Spośród metod unikania defektów wymieniono przede wszystkim metody formalne, jak rachunek CCS i CSP (ang. *calculus of communicating systems, communicating sequential processes*), sieci Petriego, logika temporalna, metoda VDM lub Z, oraz – metody systematyczne (metoda Jacksona, MASCOT – ang. *modular approach to software construction, operation and test*) i programowanie strukturalne. Jako jedną z metod do wykrywania defektów omówiono makietowanie (ang. *prototyping*), a do wykrywania uszkodzeń – asercje. Spośród głównych metod neutralizowania uszkodzeń opisano m. in. programowanie defensywne, tolerowanie defektów, wprowadzanie zróżnicowania (ang. *diversity*) i nadmiarowości (ang. *redundancy*).

KODOWANIE – PRZYKŁAD CZYNNOŚCI PROCESU IMPLEMENTOWANIA

Jedną z najważniejszych czynności implementacyjnych, oprócz kilku innych wymienionych na wstępie tego artykułu, jest kodowanie. Wbrew pozorom jednak, choć jest to bez wątpienia najstarsza i najlepiej poznana czynność wykonywana przez programistów, wcale niełatwo znaleźć sensowne standardy czy zalecenia dotyczące jej przebiegu. Jednym z ciekawszych, choć bardzo ograniczonym, bo mającym zastosowanie tylko do Fortranu 77, jest jeden z rozdziałów raportu [2]. Celem tych zaleceń jest, jak podają autorzy, uzyskanie maksymalnej przenośności, niezawodności i zrozumiałości wytworzonego kodu, a także – jego użyteczności i testowalności. Jest oczywiste, że nie wystarczy do tego norma samego języka, gdyż nieumiejętne stosowanie niektórych konstrukcji może być bardzo niebezpieczne. Konieczne są więc zalecenia dotyczące stylu programowania i taką rolę odgrywa ten fragment wymienionego raportu.

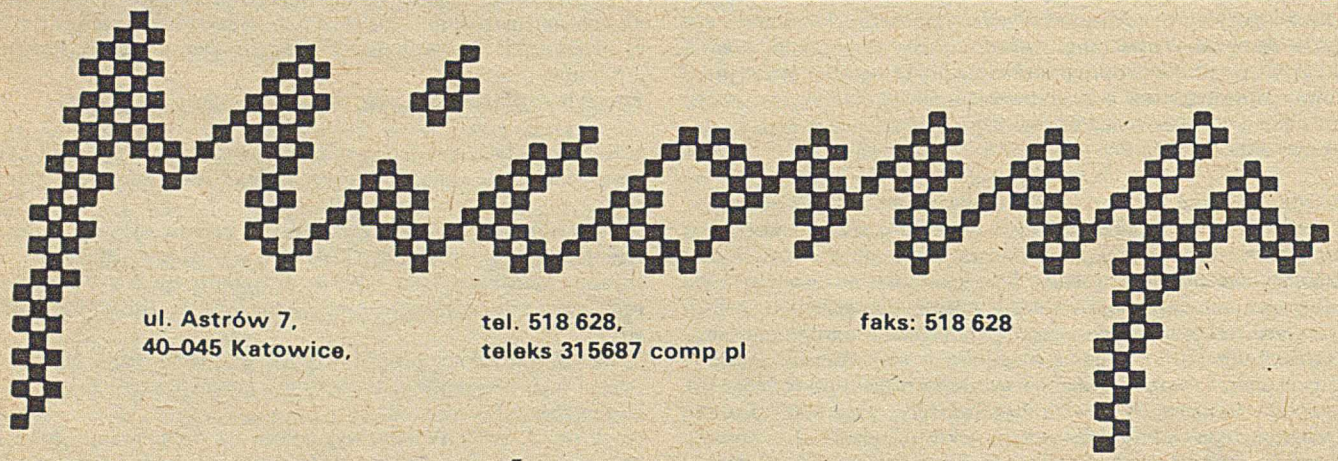
Zalecenia ogólne, jak np. to, że każdy moduł wymaga dokładnego opisu w postaci szczegółowego komentarza w nagłówku, lub że wszystkie zmienne powinny być zadeklarowane i zdefiniowane w programie głównym i w modułach, gdzie są używane itp., nie są zbyt konstruktywne. O wiele bardziej interesujące są szczegółowe wskazówki dotyczące kolejno:

- budowy modułów, włącznie z określeniem ich maksymalnego rozmiaru, organizacji programu głównego, funkcji, podprogramów i bloków, komentarzy, zasad stosowania odstępów i wcięć, wierszy kontynuacji,
- użycia struktur sterujących (wśród nich instrukcji GOTO), włącznie z numerowaniem wierszy, instrukcją CONTINUE, powrotami z modułów i obsługą błędów,
- tworzenia zmiennych i struktur danych, włącznie z ich typizacją, inicjowaniem oraz używaniem zmiennych wspólnych i lokalnych,
- wykorzystania wejścia-wyjścia i kontroli błędów.

Warto też dodać, że oprócz kodu wynikiem tej czynności powinien być raport nt. użytych narzędzi i technik (podlegający zresztą zatwierdzeniu przez zleceniodawcę).

LITERATURA

[1] Bishop P. (Ed.): Safety Assessment and Design of Industrial Computer Systems – Techniques Directory. Elsevier, Amsterdam (w druku)
[2] EPRI: Software Development and Maintenance Guidelines. Vol. 1. Topical Report EL-3089. Electrical Power Research Institute, Palo Alto (CA), May 1983
[3] Hoare C.A.R.: An Overview of Some Formal Methods for Program Design. IEEE Computer. Vol. 20, No. 9, pp. 85–91, September 1987
[4] IEEE Guide to Software Requirements Specification. IEEE Std 830–1984
[5] IEEE Recomended Practice for Software Design Descriptions. IEEE Std 1016–1987
[6] IEEE Standard for Software Life Cycle Processes. IEEE Project 1074
[7] Lamport L.: Uproszczone specyfikowanie systemów współbieżnych. Informatyka, nr 10, 11–12, 1988; nr 1, 1989
[8] Ludewig J.: Languages, Methods, and Tools for Software Specification. Proc. Conf. on Hardware and Software for Real Time Process Control (J. Zalewski, W. Ehrenberger, Eds.), pp. 225–256. North-Holland, Amsterdam, 1989
[9] Redmill F. (Ed.): Dependability of Critical Computer Systems. Vol. 1. Elsevier Applied Science, London, 1988
[10] Redmill F. (Ed.): Dependability of Critical Computer Systems. Vol. 2. Elsevier Applied Science, London, 1989
[11] Turski W. M.: Kanoniczny krok procesu programowania. Informatyka, nr 8, 9, 1988.



ZAKŁAD SYSTEMÓW MIKROKOMPUTEROWYCH

Systemy teletransmisji ODRA/ICL 1900, 2900, 39, ME 29:

- skaner MPXSCAN-8087
- procesor sieci teletransmisji danych MICOMP 8075 (emulacja ICL 750)
- program teletransmisji danych MICROSS-FXBM (IBM PC, rozproszone bazy danych, wersja sieciowa)
- adaptery, testery

Kilkadziesiąt instalacji w największych systemach na terenie całego kraju!

Integracja systemów ICL/ODRA – PC, NOVELL, XENIX

Systemy wielodostępne: supermikrokomputery ALR + UNIX!

EO/1124/89

ZBIGNIEW HUZAR, LUDWIK KUŹNIARZ
Politechnika Wrocławska

Język Lotos (2)

Specyfikacje procesów

W poprzedniej części artykułu opisano mechanizm specyfikacji typów danych, w obecnej – przedstawiono specyfikacje behawioralne systemów reaktywnych [2]. Skupiono się przy tym tylko na podstawowych mechanizmach służących do definiowania wzorców zachowań procesów, czyli tzw. wyrażeń behawioralnych. Zakłada się, że używane typy danych (rodzaje i operatory [2]) mają oczywistą interpretację. Wprowadzane pojęcia nie są definiowane formalnie, a zamiast tego podaje się przykłady ilustrujące ich znaczenie.

DEFINICJE I EGZEMPLARZE PROCESÓW

W języku oddzielnie definiuje się procesy, jako wzorce pewnej klasy obiektów, oraz oddzielnie tworzy się egzemplarze procesów – jako konkretne elementy tej klasy obiektów. Specyfikacja w Lotosie dopuszcza dynamiczne tworzenie procesów. Definicja procesu jest następująca:

```
process Nazwa_procesu  Lista_bramek_formalnych
        Lista_parametrów_formalnych : Funkcjonalność :=
    Wyrażenie_behawioralne
  where
    Definicje_lokalne
endproc
```

Nagłówek definicji procesu, zawierający jego nazwę i parametry, jest oddzielony od treści procesu znakiem „:=".

Pierwszą część parametrów procesu stanowi *Lista_bramek_formalnych*, zapisywana w postaci [*g*1, ..., *gn*], przy czym *g*1, ..., *gn* są nazwami bramek, przez które proces może komunikować się ze swoim otoczeniem. *Lista_parametrów_formalnych* jest zapisana w postaci (*x*1:*t*1, ..., *xm*:*tm*), przy czym *x*1, ..., *xm* są identyfikatorami parametrów wartościowych (odpowiednikami parametrów przekazywanych przez wartość w definicji procedury), a *xt*, ..., *tm* są rodzajami odpowiadających im parametrów. Parametry te służą nadawaniu początkowych wartości lokalnym zmiennym procesu. Trzecia część parametrów procesu – Funkcjonalność – dostarcza ogólnych informacji o sposobie kończenia działania procesu. Proces może nigdy nie zakończyć swego działania bądź zakończyć je przejściem do całkowitej bezczynności – tę sytuację określa słowo zastrzeżone **noexit**. Możliwa jest także sytuacja, w której proces kończąc swoje działanie pobudza równocześnie do działania inny proces – wtedy określa to słowo zastrzeżone **exit** z ewentualnymi parametrami. Pierwsza i druga część parametrów procesu są częściami opcjonalnymi, trzecia jest obligatoryjna. Pierwsza lista parametrów aktualnych jest komunikowana przez nazwę, a druga – przez wartość. Bramki formalne oraz parametry wartościowe są, oczywiście, obiektami globalnymi procesu i, o ile nie są przesłonięte innymi konstrukcjami, mają znaczenie ustalone w nagłówku procesu.

Wyrażenia behawioralne stanowiące treść procesów określają zbiór ciągów interakcji, w których procesy te mogą uczestniczyć – oddziaływać ze swoim otoczeniem.

Tworzenie konkretnego egzemplarza procesu polega na odwołaniu się do wybranej definicji z podaniem aktualnych wartości parametrów:

Nazwa_procesu Lista_aktualnych_bramek Lista_aktualnych_wartości

Efektem użycia takiego wywołania procesu jest utworzenie pewnego obiektu, który będzie zachowywał się zgodnie z treścią zawartą w definicji procesu, przy czym treść ta będzie zmodyfikowana stosownie do wartości parametrów aktualnych.

KOMUNIKACJA MIĘDZY PROCESAMI

Komunikacja, jaką mogą prowadzić ze sobą procesy jest w Lotosie zróżnicowana ze względu na liczbę jednocześnie komunikujących się ze sobą procesów oraz kierunki jednoczesnego przesyłania informacji. W najprostszym przypadku komunikacja odbywa się między parą procesów i polega na przesłaniu, przez wspólną bramkę, pewnego komunikatu od jednego do drugiego z tych procesów. Przesłanie komunikatu wymaga synchronizacji procesów, co oznacza, że może to nastąpić tylko wówczas, gdy w tej samej chwili jeden z nich jest gotowy do wysłania, a drugi do odbioru informacji. W pojedynczej komunikacji może uczestniczyć dowolna liczba procesów, przy czym dany proces może pewną informację wysyłać i równocześnie pewną informację odbierać.

Do prowadzenia komunikacji służą tzw. *akcje*. Wyróżnia się dwie akcje elementarne[1] *g*:*E* oraz *g*?*x*:*t*, przy czym *g* oznacza nazwę bramki, *E* jest pewnym wyrażeniem, *x* – zmienną, a *t* – identyfikatorem rodzaju. Zasięgiem zmiennej *x* jest proces, w którym wystąpiła akcja *g*?*x*:*t*, oprócz tych fragmentów, w których nastąpiło przesłonięcie zmiennej przez inny wewnętrzny proces. Zmienna *x* musi być oczywiście różna od parametrów formalnych procesu, w którym występuje.

Pierwsza z tych akcji, nazywana elementarną akcją wyjścia, polega na wysłaniu przez proces realizujący tę akcję wartości wyrażenia *E* przez bramkę *g*. Druga z tych akcji, nazywana elementarną akcją wejścia, polega na odbiorze przez proces realizujący tę akcję pewnej wartości rodzaju *t* z bramki *g* i przypisaniu tej wartości zmiennej *x*.

Najprostszy przypadek komunikacji, w której uczestniczą dwa procesy wymaga więc, aby jeden z nich był gotowy do realizacji elementarnej akcji wejścia, a drugi – elementarnej akcji wyjścia. Akcje te muszą być zgodne, tzn. obie muszą odnosić się do tej samej bramki, a wyrażenie *E* w akcji wyjścia musi być takiego samego rodzaju jak zmienna *x* w akcji wejścia. Realizację pary zgodnych akcji nazywa się interakcją.

Tabela 1. Komunikacja pary procesów

Akcja procesu pierwszego	Akcja procesu drugiego	Warunki zgodności akcji	Efekt interakcji
g!*E*1	*g*?*x*2:*t*2	*E*1 jest rodzaju *t*2	Synchronizacja oraz *x*2 := *E*1
g?*x*1:*t*1	*g*!*E*2	*E*2 jest rodzaju *t*1	Synchronizacja oraz *x*1 := *E*2
g!*E*1	*g*!*E*2	*E*1, *E*2 mają tę samą wartość	Synchronizacja
g?*x*1:*t*1	*g*?*x*2:*t*2	Rodzaje *t*1, *t*2 są identyczne	Synchronizacja oraz *x*1 := *v*, *x*2 := *v* przy czym *v* ma dowolną wartość rodzaju *t*1

W bardziej ogólnym przypadku komunikacja pary procesów może polegać na realizacji dowolnego zestawu akcji elementarnych (tabele 1). Dwie pierwsze sytuacje ujęte w tabeli 1 odpowiadają omówionemu przesłaniu wartości między dwoma procesami. W trzeciej procesy nie przesyłają między sobą żadnych wartości, a tylko synchronizują się

Praca została wykonana w ramach Centralnego Programu Badawczo-Rozwojowego 8.13 „Budowa Krajowej Akademickiej Sieci Komputerowej", cel poznawczy „Formalne metody specyfikacji i konstruowania oprogramowania sieciowego".

1) Oprócz akcji elementarnych można w Lotosie definiować również nie omawiane tu akcje strukturalne, służące do przekazywania zbioru wartości (a nie pojedynczej wartości), a także akcje i predykaty selekcjonujące.

wzajemnie. Czwarta sytuacja odpowiada interakcji, której efektem jest niedeterministyczne wygenerowanie pewnej wartości wspólnie zapamiętanej przez oba procesy. Jak już wspomniano, Lotos dopuszcza interakcje, w których bierze udział większa liczba procesów. Zbiór akcji *a*1, ..., *an* postaci:

g di

gdzie $i = 1, ..., n$, jest zbiorem zgodnych, jeżeli każde dwie akcje *g di* oraz *g dj*, dla $i,j = 1, ..., n$, są parami zgodne.

Tabela 2. Przykłady zbiorów akcji i interakcji

Akcje zgodne	Interakcje
g?x:integer *g*!8 *g*!(2+6)	Tylko jedna: *g* 8
*g?x*1:*integer* *g?x*2:*integer* *g?x*3:*integer*	Nieskończenie wiele, np.: *g* 100 *g* 123
g ?*x*:*integer* *g* !(4+8) *g* !8	Brak interakcji

W tabeli 2 przedstawiono przykłady zbiorów akcji i odpowiadających im interakcji. Identyfikator rodzaju integer oraz operator „+" należy traktować tylko jako odwołanie do oczywistych intuicji, nie należy natomiast utożsamiać ich z jakąkolwiek konstrukcją Lotosu. Podobne nieformalne odwołania się to tego rodzaju wystąpią też w dalszych przykładach.

Oprócz opisanych wyżej akcji, wyróżnia się w Lotosie jeszcze jedną akcję, nazywaną akcją wewnętrzną, oznaczaną symbolem I. Realizacja tej akcji w danym procesie nie wymaga żadnej interakcji z udziałem innego procesu. Akcja zachodzi więc automatycznie na skutek decyzji pewnego obiektu wewnętrznego danego procesu. Dla zewnętrznego obserwatora akcji *I* stanowi ona tylko informację o tym, że we wnętrzu procesu nastąpiła jakaś zmiana – komunikacja pomiędzy jego wewnętrznymi procesami.

Pojęcie akcji jest kluczowe do wyjaśnienia ogólnego schematu działania procesów. Proces rozpoczyna swoje działanie w pewnej konfiguracji początkowej, którą nieformalnie należy rozumieć jako wyczerpujący zestaw informacji o stanie zmiennych procesu oraz o położeniu sterowania wewnątrz procesu. Z położenia sterowania wewnątrz procesu wynika zbiór akcji procesu, w którym może on w danej chwili uczestniczyć. Wykonanie jednej z akcji tego zbioru zależy od uzgodnienia procesu z otoczeniem. Gdy nastąpi takie uzgodnienie (tzn. synchronizacja zgodnych akcji), wówczas zachodzi interakcja procesu z otoczeniem, której efektem, oprócz przesłania odpowiednich informacji, jest zmiana konfiguracji procesu. W nowej konfiguracji proces staje się ponownie gotowy do wykonania pewnej akcji, spośród nowego zbioru dopuszczalnych akcji. Zatem działanie procesu przebiega zgodnie ze schematem:

pierwotna konfiguracja
|
interakcja
↓
nowa konfiguracja

Działanie takie może przebiegać nieskończenie długo bądź też może zakończyć się po skończonej liczbie interakcji osiągnięciem konfiguracji końcowej albo konfiguracji blokującej. W konfiguracji końcowej zbiór akcji oferowanych przez proces do wykonania jest zbiorem pustym. Konfiguracja blokująca zawiera niepusty zbiór akcji do wykonania, lecz żadna spośród tych akcji nie jest zgodna z żądaną akcją, w której gotowe jest uczestniczyć otoczenie procesu.

Jeżeli $k1, k2, ...$ oznacza konfiguracje procesu, a $\lambda1, \lambda2, ...$ – interakcje, to obliczeniem procesu nazywa się ciąg:

$k1 -\lambda1 \rightarrow k2 -\lambda2 \rightarrow k3 -\lambda3 \rightarrow ...,$
a $\langle \lambda1, \lambda2, ... \rangle$
jest ciągiem interakcji towarzyszących temu obliczeniu.

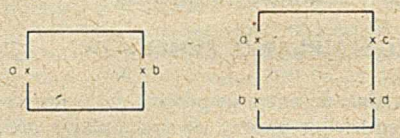
PODSTAWOWE WYRAŻENIA BEHAWIORALNE

Wyrażenia behawioralne stanowią treści procesów i opisują ich działania, tzn. definiują zbiory ciągów interakcji, które procesy mogą prowadzić ze swym otoczeniem. Ze składniowego punktu widzenia wyrażenie jest zbudowane z akcji oraz z atomowych (elementarnych) wyrażeń behawioralnych, połączonych ze sobą odpowiednimi konstruktorami.

Proces pusty, prefiksowanie akcją

Najprostszym wyrażeniem elementarnym jest proces pusty **stop**. Charakteryzuje się on pustym zbiorem akcji do wykonania. Natomiast najbardziej elementarnym konstruktorem jest konstruktor prefiksowania akcją, oznaczany symbolem „;". Jeżeli *a* jest akcją, a *B* jest pewnym wyrażeniem behawioralnym, to *a;B* jest nowym wyrażeniem, zwanym wyrażeniem prefiksowanym akcją *a*. Opisuje ono zbiór takich ciągów interakcji, których pierwszym elementem jest interakcja wynikająca z realizacji *a*, pozostałe zaś elementy są ciągami interakcji reprezentowanymi przez *B*.

Przykład 1



Rys. 1. Schematy do przykładów 1–4

Bufor (rys. 1 a), który działa w taki sposób, że jednokrotnie wczytuje z bramki *we* pewną liczbę, a następnie wyprowadza tę liczbę przez bramkę *wy*, po czym kończy działanie, można opisać jako następujący proces:

process *Bufor* [*we, wy*] : **noexit** : =
*we?x:integer;wy!x;***stop**
endproc

Przykładowe obliczenie egzemplarza tego procesu, utworzonego w wyniku wywołania *Bufor* [*a,b*], może być następujące:

a?x:integer;b!x:**stop** – *konfiguracja początkowa*
|
a 10 – *pierwsza interakcja*
↓
b! 10;**stop** – *konfiguracja pośrednia*
|
b 10 – *druga interakcja*
↓
stop – *konfiguracja końcowa*

Przykład uściśla pojęcie konfiguracji. Konfigurację procesu reprezentuje wyrażenie behawioralne. Początkową konfigurację procesu reprezentuje wyrażenie behawioralne stanowiące treść definicji procesu, zmodyfikowane odpowiednim zastąpieniem parametrów formalnych parametrami aktualnymi. W omawianym przykładzie dotyczy to tylko nazw bramek. Po każdej interakcji początkowe wyrażenie behawioralne jest przekształcane na nowe wyrażenie, które ponadto jest modyfikowane w ten sposób, że zmienna, której podczas interakcji przypisano pewną wartość, zostaje w nowym wyrażeniu zastąpiona tą wartością. W przykładowym obliczeniu dotyczy to zmiennej *x* w pośredniej konfiguracji – zamiast akcji *b!x* występuje tam *b*!10. Obliczenie jest skończone, gdyż osiąga konfigurację końcową reprezentowaną przez proces pusty **stop**.

Rekurencja

Bufor opisany w poprzednim punkcie stanowi mało ciekawy obiekt. Bardziej realistyczny byłby bufor, który czynności wpisywania i odczytywania przechowywanej zawartości mógłby wykonywać nieskończenie wiele razy. Przedstawienie takiego bufora wymaga użycia mechanizmu rekurencji, tzn. wywoływania w treści danego procesu nowego egzemplarza tego samego typu procesu.

Przykład 2

process *Buf* [*we, wy*] : **noexit** :=
we?*x*:*integer*;*wy*!*x*; *Buf* [*we, wy*]
endproc

Obliczenie egzemplarza *Buf* [*a, b*] takiego typu procesu jest oczywiście nieskończone, a jego przykładowy fragment może mieć postać:

a?*x*:*integer*; *b*!*x*; *Buf* [*a, b*]
|
a 20
↓
b!20; *Buf* [*a, b*]
|
b 20
↓
a?*x*:*integer*; *b*!*x*; *Buf* [*a, b*]
|
a 13

Przykład 3

Nową wersją procesu *Buf* będzie proces zdefiniowany poniżej. Różni się on od procesu *Buf* tym, że generując egzemplarz procesu zadaje się początkową zawartość bufora. W tym celu wykorzystuje się mechanizm parametrów formalnych służących przekazywaniu wartości.

process *Buf*1 [*we, wy*] (*x*:*integer*) : **noexit** :=
wy!*x*; *we*?*y*:*integer*; *Buf*1 [*we, wy*] (*y*)
endproc

Obliczenie procesu *Buf*1 [*a, b*] (10) łatwo wyznaczyć opierając się na poprzednim przykładzie. Należy tylko zwrócić uwagę na to, że utworzenie egzemplarza procesu, z konkretną wartością parametru wartościowego *x*, spowoduje zastąpienie tego parametru w wyrażeniu behawioralnym, stanowiącym treść procesu, konkretną wartością aktualną – liczbą 10.

Wybór

Podstawowym konstruktorem umożliwiającym utworzenie zbioru akcji, jednocześnie oferowanych do realizacji przez proces, jest konstruktor wyboru, oznaczany symbolem □. Jeżeli *B*1, *B*2 są wyrażeniami behawioralnymi, to *B*1□ *B*2 jest nowym wyrażeniem, którego znaczeniem jest zbiór ciągów interakcji stanowiący mnogościową sumę zbiorów ciągów interakcji odpowiadających znaczeniu *B*1 oraz *B*2. To, czy proces opisany wyrażeniem *B*1□ *B*2 będzie zachowywał się zgodnie ze schematem *B*1, czy zgodnie ze schematem *B*2, zależy od uzgodnienia z otoczeniem. Jeżeli pierwsza wybrana do wykonania akcja należy do *B*1, to dalsze zachowanie procesu będzie przebiegać według *B*1, a jeżeli należy do *B*2 – to według *B*2. Wybór ten jest, jak widać, niedeterministyczny.

Przykład 4

Rozpatrzmy bufor dwupozycyjny, stanowiący połączenie dwóch równocześnie pracujących pojedynczych buforów.

Process *podwójny_buf* [*we*1, *we*2, *wy*1, *wy*2] : *noexit* :=
*we*1?*x*:*integer*; *wy*1!*x*; *Podwójny_buf* [*we*1, *we*2, *wy*1, *wy*2]
[ ]*we*2?*y*:*integer*; *wy*2!*y*; *Podwójny_buf* [*we*1, *we*2, *wy*1, *wy*2]
endproc

Obliczenia procesu są oczywiście nieskończone, a przykładowy ich fragment dla egzemplarza procesu *Podwójny_buf* [*a, b, c, d*] (rys. 1 b) przedstawiono poniżej:

a?*x*:*integer*; *b*!*x*; *Podwójny_buf* [*a, b, c, d*]
[ ] *c*?*y*:*integer*; *d*!*y*; *Podwójny_buf* [*a, b, c, d*]
|
a 19 – *Możliwość interakcji z udziałem bramki c*
↓
b!19; *Podwójny_buf* [*a, b, c, d*]
|
b 19
↓
a?*x*:*integer*; *b*!*x*; *Podwójny_buf* [*a, b, c, d*]
[ ] *c*?*y*:*integer*; *d*!*y*; *Podwójny_buf* [*a, b, c, d*]
|
c 12 – *Możliwość interakcji z udziałem bramki a*
↓
d!12; *Podwójny_buf* [*a, b, c, d*]
|
d 12

W obliczeniu zaznaczono konfiguracje, po których mogą zajść interakcje z udziałem innych bramek niż wskazane. Należy jednak zauważyć, że przedstawiony proces ma istotne ograniczenie: jeżeli jako pierwsza nastąpi interakcja z udziałem bramki *a*, to następną może być tylko interakcja z udziałem bramki *b*, chociaż oczekiwałoby się także możliwości interakcji z udziałem bramki *c*.

Złożenie równoległe

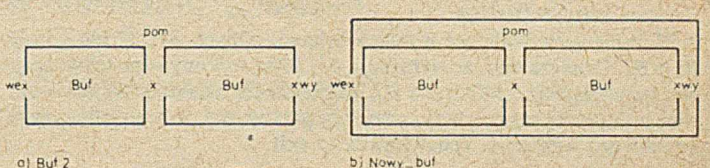
Konstruktor równoległego złożenia wyrażeń behawioralnych *B*1, *B*2 ma postać *B*1 | [*g*1, ... *gn*] | *B*2, przy czym *g*1, ..., *gn* są nazwami bramek. Interpretacja powstałego wyrażenia jest następująca. Wyrażenia *B*1 i *B*2 reprezentują dwa obiekty działające równolegle. Obiekty te prowadzą komunikację ze swoim otoczeniem wykorzystując swoje bramki, wśród których mogą być również bramki *g*1, ..., *gn*, będące jedynymi bramkami do komunikacji między obu obiektami. Oznacza to, że każda interakcja angażująca jedną z tych bramek zachodzi tylko wówczas, gdy biorą w niej udział oba obiekty. Dokładniej, jeżeli taka bramka jest bramką wewnętrzną, tzn. służy wyłącznie do komunikacji między tymi obiektami, to w interakcji z jej udziałem uczestniczą tylko te dwa obiekty. Jeżeli natomiast jest bramką zewnętrzną, tzn. służy również do komunikacji z otoczeniem tych obiektów, to w interakcji z jej udziałem uczestniczą równocześnie oba obiekty oraz ich otoczenie.

Przykład 5

Wykorzystując jeden z poprzednich przykładów można zbudować bufor służący tworzeniu dwuelementowych kolejek.

```
process Buf2 [we,pom,wy] : noexit :=
   Buf [we,pom] |[pom]| Buf [pom,wy]
   where
      process Buf [we,wy] : noexit :=
         we?x:integer; wy!x; Buf [we,wy]
      endproc
endproc
```

W przykładzie tym, oprócz wprowadzenia równoległego złożenia procesów, wykorzystano także możliwość definiowania lokalnych obiektów – w tym przypadku jest to definicja procesu *Buf*.



Rys. 2. Schematy do przykładów 5 i 6

W omawianym przykładzie rola bramki *pom* wydaje się nieco sztuczna – rys. 2. Intuicyjnie oczekiwałoby się, że ma ona pełnić tylko funkcję pomocniczą służąc wyłącznie komunikacji między dwoma egzemplarzami procesu *Buf*. Tymczasem z definicji procesu *Buf* 2 wynika, że służy ona także do komunikacji z otoczeniem tego procesu. Zatem każdorazowe przesłanie wartości z procesu *Buf* [*a, p*], przez bramkę *P* do procesu *Buf* [*p, b*], wymaga synchronizacji obu procesów z otoczeniem procesu. Do przesłaniania komunikacji zachodzących między wewnętrznymi bramkami procesu służy konstrukcja postaci:

hide *g*1, ... *gn* **in** *B*

przy czym *g*1, ... *gn* są przesłanianymi bramkami w wyrażeniu *B*.

Przykład 6

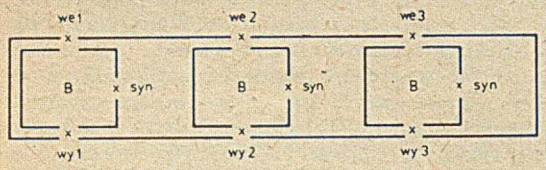
Poprzedni przykład zmodyfikowany w myśl powyższych uwag – rys. 2b – ma postać:

```
process Nowy_buf2 [we,wy] : noexit :=
   hide pom in
      Buf [we,pom] |[pom]| Buf [pom,wy]
      where
         process Buf [we,wy] : noexit :=
            wy?x:integer; we!x; Buf [we,wy]
         endproc
endproc
```

Obliczenia tego procesu będą wyglądać podobnie jak poprzednio, z tym, że interakcje z udziałem bramki *pom* nie będą dostrzegane z otoczenia procesu (dokładniej, będą traktowane jako pewna akcja wewnętrzna I) oraz interakcje z jej udziałem nie będą wymagać udziału otoczenia.

Przykład 7

Ilustracją interakcji, w których uczestniczą więcej niż dwa procesy, jest działanie systemu, składającego się z trzech równoległych procesów (rys. 3)



Rys. 3. Schemat do przykładu 7

Każdy z nich odczytuje ze swojej bramki wejściowej liczbę, którą następnie wyprowadza na swoją bramkę wyjściową. Warunkiem wyprowadzenia liczby przez dany proces jest to, aby wszystkie pozostałe procesy pobrały już liczby ze swych wejść. W celu spełnienia tego warunku wprowadza się bramkę synchronizującą *syn*, przez którą nie przesyła się żadnej informacji, lecz interakcja z jej udziałem wymaga uczestnictwa trzech wymienionych procesów.

```
process Rejestr [we1,we2,we3,wy1,wy2,wy3] : noexit :=
hide syn in
   B[we1,wy1,syn] |[syn]| B[we2,wy2,syn] |[syn]| B[we3,wy3,syn]
where
   process B[we,wy,syn] : noexit :=
      we?x:integer; syn; wy!x; B[we,wy,syn]
   endproc
endproc
```

Złożenie uaktywniające

Złożenie uaktywniające wyrażeń behawioralnych $B1$, $B2$ ma postać $B1 \gg B2$. Oznacza ono, że wyrażenia $B1$, $B2$ są wykonywane sekwencyjnie: najpierw $B1$, a następnie $B2$. Zakończenie działania $B1$ nie może więc polegać na zaprzestaniu wszelkiej aktywności, lecz na osiągnięciu końcowego wyrażenia behawioralnego **exit**.

Przykład 8

Automat do sprzedawania biletów pracuje cyklicznie, wykonując na przemian pobieranie pieniędzy i wydawanie biletów:

```
process Automat[moneta,bilet] : noexit :=
   Pobierz_należność[moneta] >> Wydaj_bilet[bilet]
   >> Automat[moneta,bilet]
where
   process Pobierz_należność[mon] : exit :=
      mon; exit
   endproc
   process Wydaj_bilet[bil] : exit :=
      bil; exit
   endproc
endproc
```

A oto przykładowy fragment obliczenia procesu *Automat* [*m*, *b*]:

Pobierz_należność[*m*]≫ *Wydaj_bilet*[*b*]≫ *Automat*[*m*, *b*]

|

m

↓

exit≫ *Wydaj_bilet*[*b*]≫ *Automat*[*m*, *b*]

|

I

↓

Wydaj_bilet[*b*]≫ *Automat*[*m*, *b*]

|

b

Jak widać, wykonaniu wyrażenia **exit** towarzyszy zajście akcji wewnętrznej I. Należy też zwrócić uwagę na to, że funkcjonalność procesów *Pobierz_należność* i *Wydaj_bilet* – a więc procesów, które mogą przekazywać sterowanie swoim następnikom – jest określona słowem zastrzeżonym **exit** w ich nagłówkach.

SEMANTYKA

Znaczeniem, czyli semantyką procesu w Lotosie jest zbiór ciągów interakcji, w których – z punktu widzenia zewnętrznego obserwatora – proces ten może uczestniczyć. Jak łatwo przekonać się na podstawie przykładów, każde obliczenie procesu wyznacza jednoznacznie pewien ciąg interakcji, w których proces uczestniczy. Zatem posiadając mechanizm wyznaczania obliczeń procesu posiada się tym samym mechanizm generowania ciągów interakcji. Takie podejście zastosowano w definicji semantyki procesów w Lotosie. Jest to więc podejście operacyjne, oparte na tzw. etykietowanym systemie przejść.

Formalnie etykietowanym systemem przejść nazywa się czwórkę $\langle S, A, T, s_0 \rangle$, przy czym: S jest niepustym zbiorem konfiguracji; A jest zbiorem interakcji; T jest zbiorem relacji przejść, który dla każdego elementu $a \in A$ zawiera dokładnie jedną relację $-a\rightarrow \in S \times S$; $s_0 \in S$ jest początkową konfiguracją. Intuicyjnie, pierwsze dwa elementy etykietowanego systemu przejść będą miały interpretację używaną dotychczas, a więc konfiguracje będą reprezentowane przez pewne wyrażenia behawioralne, a interakcje będą wyznaczane przez bramki ze skojarzonym zbiorem przesyłanych przez nie wartości. Podobnie początkową konfiguracją będzie pewne wyrażenie behawioralne. Zbiór wszystkich konfiguracji będzie generowany przez składnię języka Lotos

Tabela 3. Zestaw aksjomatów i reguł związanych ze znaczeniem konstruktorów

Konstrukcja	Aksjomaty i reguły
stop	Brak
a;B	$a;B\text{-}a\longrightarrow B$
B1[]B2	$\dfrac{B1\text{-}a\longrightarrow B1'}{B1[]B2\text{-}a\longrightarrow B1'}$ $\qquad$ $\dfrac{B2\text{-}a\longrightarrow B2'}{B1[]B2\text{-}a\longrightarrow B2'}$
hide g in B	$\dfrac{B\text{-}a\longrightarrow B',\ a\neq g}{(\text{hide g in B})\text{-}a\longrightarrow (\text{hide g in B}')}$ $\qquad$ $\dfrac{B\text{-}g\longrightarrow B'}{(\text{hide g in B})\text{-}I\longrightarrow (\text{hide g in B}')}$
B1 |[g]| B2	$\dfrac{B1\text{-}a\longrightarrow B1',\ a\neq g}{B1\ |[g]|\ B2\text{-}a\longrightarrow B1'\ |[g]|\ B2}$ $\qquad$ $\dfrac{B2\text{-}a\longrightarrow B2',\ a\neq g}{B1\ |[g]|\ B2\text{-}a\longrightarrow B1\ |[g]|\ B2'}$ $\qquad$ $\dfrac{B1\text{-}g\longrightarrow B1',\ B2\text{-}g\longrightarrow B2'}{B1\ |[g]|\ B2\text{-}I\longrightarrow B1'\ |[g]|\ B2'}$
B1 >> B2	$\dfrac{B1\text{-}a\longrightarrow B1',\ B1 \neq \text{exit}}{B1>>B2\text{-}a\longrightarrow B1'>>B2}$ $\qquad$ $\dfrac{B1 = \text{exit}}{B1>>B2\text{-}I\longrightarrow B2}$
p[g]	$\dfrac{\text{process p[g']: ... := B endproc},\ B[g/g']\text{-}a\longrightarrow B'}{p[g]\text{-}a\longrightarrow B'}$ gdzie B[g/g'] oznacza zastąpienie bramki g', w treści B procesu p, przez bramkę g.

Zbiór relacji przejść będzie zadawany aksjomatycznie przez podanie pewnego skończonego zbioru aksjomatów i reguł wnioskowania. Zestaw aksjomatów i reguł związanych ze znaczeniem dotychczas wprowadzonych konstruktorów wyrażeń behawioralnych przedstawiono w tabeli 3.

dokończenie na str. 31

SWEDEX UNIVERSAL Co.

PRZEDSIĘBIORSTWO ZAGRANICZNE W POLSCE

Biuro: ul. Kolejowa 2, 05–250 Radzymin
Teleks 815888 swedx pl, telefon: Warszawa 762004 w. 13 lub 365
Zakład Elektroniki: ul. Wspólna 1, 05–440 Wesoła-Zielona
Teleks 813935 swedx pl, telefon: Warszawa 153365

OFERTA ZAKŁADU ELEKTRONIKI

MODEM TRANSMISJI DANYCH 1230P

- umożliwia transmisję dwukierunkową jednoczesną (full duplex) po łączach telefonicznych komutowanych i dzierżawionych: w trybie asynchronicznym z szybkościami 0–300 b/s, 600 b/s i 1200 b/s; w trybie synchronicznym z szybkością 1200 b/s;
- zgodny z zaleceniami CCITT V.21, V.22;
- posiada świadectwo homologacji Instytutu Łączności PRL.

KONCENTRATOR STANOWISK OPERATORSKICH KSO-4 DO REJESTRATORA DANYCH MERA 9150 (SEECHEK)

- pozwala na dołączenie do Mery 9150 poprzez jedno łącze transmisji danych do 4 stanowisk operatorskich i drukarki systemowej;
- nie wymaga zmian w systemie operacyjnym i sprzęcie rejestratora;
- posiada wbudowane testy pozwalające na szybką lokalizację uszkodzeń i sprawdzenie łącza transmisji danych;
- jest produkowany także w wersji z wbudowanym modemem stałoprądowym KN-9600.

ADAPTER TELEGRAFICZNY ATG-2 DO MERY 9150

- umożliwia wprowadzanie danych i teleksów bezpośrednio z dalekopisu (sieci teleksowej) do Mery 9150 z kontrolą i automatycznym przesyłaniem komunikatów (teleksów) zwrotnych.

ADAPTER TELEKOMUNIKACYJNY AT 8-MPX (UPD asynchroniczne i synchroniczne)

- pozwala na dołączenie do EMC ODRA 1305 poprzez multiplekser MPX 325 ośmiu zdalnych terminali np. drukarki DZM 180 KSRE, monitora ekranowego ICL 7181 lub odpowiednika produkcji krajowej, stacji ICL 7020, mini- lub mikrokomputerów z emulatorami powyższych terminali;
- realizuje automatyczne rozłączenie połączeń komutowanych.

INTERFEJSY POMIAROWE SU-GPIB I SU-GPIBF STANOWIĄCE PEŁNĄ IMPLEMENTACJĘ NORMY IEC-625 (IEEE 488) UMOŻLIWIAJĄCE STEROWANIE PROCESEM POMIAROWYM Z SZYBKOŚCIĄ DO 500 kB/s

- oprogramowanie: procedury wywoływane z takich języków programowania, jak C, PASCAL, FORTRAN, BASIC;
- układ interfejsu zrealizowany jest w oparciu o kontroler 7210 oraz bufory firmy Texas Instruments.

PAKIET TRANSMISJI SZEREGOWEJ SYNCHRONICZNEJ/ASYNCHRONICZNEJ BSC DO KOMPUTERÓW KOMPATYBILNYCH Z IBM PC XT/AT

- umożliwia pracę komputera jako terminala np. ICL 7181, ICL 7020, IBM 2780, IBM 3780, IBM 3270;
- dostarczamy oprogramowanie emulacyjne pakietu BSC.

PROGRAMATOR PAMIĘCI EPROM ORAZ MIKROKOMPUTERÓW JEDNOUKŁADOWYCH

- umożliwia programowanie pamięci typu 2716--27512 i ich funkcjonalnych odpowiedników o napięciach programujących 12,5; 21; 25 V oraz mikrokomputerów jednoukładowych 8741, 8748, 8749, 8751, 8755;
- automatycznie wybiera parametry programowania;
- współpracuje z komputerem przez styk RS-232 (V.24);
- oprogramowanie do komputerów IBM PC XT/AT.

KOMPUTERY SU 88-PC W PEŁNI KOMPATYBILNE Z IBM PC KONFIGUROWANE ZGODNIE Z ŻYCZENIEM KLIENTA.

EO/1263/88

Dwa kompilatory Moduli 2 dla IBM PC

Jedne z pierwszych kompilatorów języka Modula 2, jakie powstały dla mikrokomputerów IBM PC, opracowano w firmach: MRI (*Modula Research Institute*) i Logitech w USA. Oba kompilatory oparto na oryginalnej implementacji dla komputera Lilith, powstałej na Uniwersytecie w Zurychu. Podstawowa różnica między nimi polega na tym, że kompilator MRI generuje M-kod, zgodny z kodem komputera Lilith, a kompilator Logitech generuje kod maszynowy mikroprocesora 8088.

Kompilator Logitech w wersji 1.10[1] może współpracować z koprocesorem 8087, wymaga użycia dwóch napędów dysków elastycznych lub dysku stałego i praktycznie 256 KB pamięci RAM ponad to, co zajmuje DOS; współpracuje ze wszystkimi wersjami DOS-a, a także z systemem CP/M-86. Kompilator MRI działa na komputerze o dwóch napędach dysków elastycznych i 192 KB pamięci RAM. Oczywiście, oba kompilatory działają znacznie szybciej na zestawach z dyskiem elektronicznym.

Praktyczne wykorzystanie obu kompilatorów wymaga uważnego użycia plików instalacyjnych i ustalenia konfiguracji, tzn. założenia odpowiednich katalogów do przechowywania dużej liczby plików, dobrania odpowiednich parametrów w pliku CONFIG.SYS oraz ustalenia właściwych opcji na poziomie systemowym w pliku AUTOEXEC.BAT. Żaden z kompilatorów nie działa bezpośrednio w systemie PC-DOS ani nie generuje bezpośrednio wykonywalnego kodu. System MRI zawiera własną powłokę, czyli interpreter poleceń nadbudowany nad DOS. Kompilator i utworzone programy są wywoływane przez tę powłokę. System Logitech, choć nie ma powłoki, zawiera podsystem wykonawczy (ang. *runtime system*), który jest wzywany przez kompilator i programy użytkowe. Żaden z omawianych systemów nie ma natomiast własnego edytora.

Wywołanie powłoki MRI następuje po wydaniu polecenia INTERP, po czym można wywołać kompilator poleceniem MODULA lub wykonywać skompilowane programy. Wyjście z powłoki wykonuje się klawiszem CTRL-C. Kompilator Logitech wywołuje się poleceniem M2 COMP. M2 jest to nazwa systemu wykonawczego, która musi poprzedzać nazwę każdego programu wykonywanego pod kontrolą systemu PC-DOS.

Po wywołaniu obu systemów dalsze postępowanie jest prawie identyczne, np. dla kompilatora Logitech:

```
M2 COMP
  source file:
przebieg 1      Syntax Analysis
  ⟨wykaz dołączanych modułów⟩
przebieg 2      Declaration Analysis
przebieg 3      Block Analysis
przebieg 4      Code Generation
  ⟨informacja końcowa⟩
```

Podobieństwo konwersacji w obu systemach jest skutkiem faktu, że oparto je na kompilatorze Moduli 2 powstałym na Politechnice w Zurychu. W szczególności czteroprzebiegowa struktura kompilatora jest pozostałością jego pierwotnej wersji zrealizowanej na komputerze PDP-11 o bardzo małej przestrzeni adresowej.

Program skompilowany przez kompilator Logitech należy przed wykonaniem poddać jeszcze konsolidacji, czego nie wymaga już system MRI. Wywołanie konsolidatora, który łączy oddzielnie skompilowane części programu, jest podobne jak kompilatora M2 LINK. Z kolei, interpreter MRI ma bardzo długi czas ładowania do pamięci operacyjnej (nawet z dysku stałego), co może być bardziej kłopotliwe przy częstym opuszczaniu go, na przykład w celu poprawienia błędów, i ponownym wywołaniu. Z tego powodu zawarto w nim możliwość bezpośredniego wywołania interpretera DOS-a COMMAND.COM, przez poprzedzenie nazwy polecenia wykrzyknikiem. Po zredagowaniu programu można wtedy powrócić do systemu MRI poleceniem EXIT. Należy jednak uważać, aby przed ponownym wejściem do systemu MRI powrócić do właściwego katalogu; w przeciwnym razie może to spowodować załamanie systemu operacyjnego.

[1] Obecnie w sprzedaży jest już trzecia wersja tego kompilatora.

Po wykonaniu kilku przykładowych programów można się zorientować, że kompilator Logitech (w wersji 1.10) jest znacznie wolniejszy[2] od kompilatora MRI. Jednakże, programy wykonywalne przetłumaczone przez ten kompilator są o wiele szybsze niż odpowiednie programy przetłumaczone przez kompilator MRI. Najprostszy program wyświetlający na ekranie tekst, *"The program worked"* (wydruk 1) jest kompilowany przez kompilator MRI w ciągu ok. 40 s, a przez kompilator Logitech – w ciągu 65 s (nie licząc czasu konsolidacji). Jeżeli w danym zastosowaniu istotny jest czas wykonania programu, to znacznie korzystniejsze będzie użycie kompilatora Logitech, który wytwarza kod wielokrotnie szybszy niż robi to system MRI. Choć skompilowanie prostego programu przedstawionego na wydruku 2 zajmuje kompilatorowi MRI 20 s, a kompilatorowi Logitech 55 s (nie licząc konsolidacji – choć byłaby ona szybka ze względu na brak modułów importowanych), to przetłumaczony program wykonuje się w pierwszym wypadku ok. 60 s, a w drugim tylko 8 s, co jest prawie sześciokrotnym zyskiem.

```
MODULE Examp1;
  FROM Terminal IMPORT WriteString, WriteLn, Read;
  VAR ch: CHAR;
BEGIN
  WriteString('The program worked! (Hit a key)');
  WriteLn;
  Read(ch);
END Examp1.
```

Wydruk 1

Warto zauważyć, że drugi program kompiluje się znacznie szybciej niż pierwszy, mimo że jego część wykonywalna jest bardziej złożona. Jest tak dlatego, że pierwszy program importuje procedurę z zewnątrz. Ta czynność pochłania czas, ponieważ kompilator musi odczytać definicję modułu i skontrolować jego zawartość zgodnie z zawartością klauzuli IMPORT. Jednakże, w praktyce prawie każdy program w Moduli 2 importuje procedury z innych modułów, choćby do wprowadzania lub wyprowadzania.

```
MODULE Examp3;
FROM InOut IMPORT WriteInt;
VAR i,j,sum : INTEGER;
BEGIN
        sum := 0;
        FOR i:=1 TO 1000 DO
                FOR j:=100 TO 1 BY -1 DO
                        sum := sum+(i-j)+(j-1);
                END;
        END;
        WriteInt(sum, 4);
END Examp3.
```

Wydruk 2

W celu dokładniejszego porównania szybkości obu systemów można wykonać znany program wzorcowy do generowania liczb pierwszych – Sito Erastotenesa. System Logitech potrzebował ok. 65 s na skompilowanie tego programu, 25 s – na skonsolidowanie i ok. 17 s – na wykonanie. W systemie MRI kompilacja trwała 55 s, a wykonanie prawie 3 minuty.

```
MODULE Examp4;
FROM RealInOut IMPORT WriteReal;
VAR x,y : ARRAY[1..100] OF REAL;
      sum : REAL;
VAR i,j : INTEGER;
BEGIN
        FOR i:=1 TO 10 DO
                x[i] := FLOAT(i);
                y[i] := FLOAT(i);
        END;
        sum := 0.0;
        FOR i:=1 TO 10 DO
                FOR j:=1 TO 10 DO
                        sum := sum + x[j]*y[j];
                        sum := sum/FLOAT(i);
                END;
        END;
        WriteReal(sum, 18);
END Examp4.
```

Wydruk 3

[2] Wszystkie ceny podano za miesięcznikiem Byte.

Reprezentacja danych w kompilatorach jest identyczna, z wyjątkiem typów rzeczywistych. W systemie MRI liczby rzeczywiste są reprezentowane jako wartości 32-bitowe, a operacje arytmetyczne na nich są wykonywane programowo. System Logitech używa 64-bitowej reprezentacji liczb rzeczywistych i może generować kod dla koprocesora arytmetycznego 8087, co jest wielką zaletą. Przykładowy program z wydruku 3 został skompilowany przez kompilator MRI w ciągu 30 s i wykonany w ciągu 25 s, a kompilacja w systemie Logitech trwała 50 s, natomiast wykonanie, z zainstalowanym koprocesorem 8087 tylko ok. 3 s.

Dyrektywy kompilacji umieszczone w programach napisanych w Moduli 2 są bardzo podobne do dyrektyw Pascala. Nazwę dyrektywy i jej argument ujmuje się w nawiasy komentarza. Kompilator Logitech ma trzy dyrektywy:

(*$R+*) – kontrola typów i zawężeń,
(*$S+*) – badanie przepełnienia stosu,
(*$T+*) – testowanie indeksów.

Każdą dyrektywę można znieczulić używając znaku minus zamiast plus lub zmienić jej znaczenie na poprzednie używając znaku równości. W systemie MRI występują tylko dyrektywy *R* i *T*. W obu systemach możliwe jest również stosowanie kilku opcji w poleceniu kompilacji, np. tworzenie wydruku, jawne nazywanie pliku z tablicą symboli itp.

Obydwa kompilatory są zgodne z raportem Moduli 2 (N. Wirth, Modula 2, WNT, Warszawa, 1987), jednak system Logitech ma dodatkowe procedury, umieszczone w module SYSTEM, umożliwiające odczytywanie i ustawianie rejestrów mikroprocesora 8088 (GETREG, SETREG), zabranianie, zezwalanie i inicjowanie przerwań (ENABLE, DISABLE, SWI), odczytywanie i zapisywanie portów procesora INBYTE, OUTBYTE, INWORD, OUTWORD), generowanie wywołań systemowych DOS (DOSCALL), wplatanie kodu asemblerowego (CODE) i in. Istnieją też m.in. następujące dodatkowe moduły biblioteczne:

ASCII – stałe symboliczne oznaczające znaki niedrukowalne;

CardinalIO – wejście-wyjście dziesiętne i szesnastkowe dla liczb porządkowych;

Conversion, NumberConv, RealConv – konwersja typów całkowitoliczbowych i porządkowych na napisy, konwersja liczb na napisy i odwrotnie, konwersja liczb zmiennoprzecinkowych na napisy i odwrotnie;

FileMessage, FileNames, Options – wyprowadzenie stanu pliku na terminal i odczytanie specyfikacji pliku (wraz z opcjami) z terminala;

Processes – programowanie współbieżne;

ProgMessage – wyprowadzanie na terminal komunikatu o stanie programu;

RS232Code, RS232Int, RS232Poll – wejście-wyjście przez port RS232 sterowane przerwaniami lub odpytywaniem;

Strings – obsługa napisów o zmiennej długości.

Faktycznie nie są to rozszerzenia języka, lecz większość użytkowników tak je traktuje. W zestawie tym brakuje jednak procedur graficznych, nawet najprostszych. System MRI zawiera natomiast tylko procedury zdefiniowane w raporcie Moduli 2, a więc programy zależne od procesora lub systemu operacyjnego realizuje się znacznie łatwiej w systemie Logitech.

System Logitech umożliwia nakładkowanie. Pamięć jest dzielona na globalny segment danych o wielkości 64 KB na dane statyczne, stos programu przeznaczony na parametry procedur, zmienne lokalne i zmienne pośrednie używane w wartościowaniu wyrażeń (maksymalnie 64 KB) i stertę użytkownika (ang. *heap*), która umożliwia tworzenie i przechowywanie dużych struktur danych podczas wykonywania programu (jej wielkość jest ograniczona jedynie pojemnością zainstalowanej pamięci). Ta ostatnia cecha jest szczególnie użyteczna przy używaniu typów nieprzenikalnych (ang. *opaque*), które są zawsze realizowane jako wskazania na pewne struktury danych.

Porównanie czasów kompilacji i wykonania programów dla obu systemów jest zrozumiałe, jeśli się zważy, że kompilator MRI tłumaczy programy na M-kod, który został zaprojektowany z myślą o efektywnej kompilacji Moduli 2. Z tego faktu wynika większa szybkość kompilatora MRI niż kompilatora Logitech. Jednakże, programy skompilowane w systemie MRI są znacznie wolniejsze, ponieważ są wykonywane przez interpretację M-kodu, natomiast programy wynikowe w systemie Logitech są tłumaczone bezpośrednio na język wewnętrzny procesora 8088 i nie wymagają interpretowania.

Użytkownicy pragnący nauczyć się Moduli 2 i pisać programy do własnego użytku będą prawdopodobnie woleli korzystać z systemu MRI. Większe znaczenie ma dla nich możliwość szybkiego kompilowania małych programów, natomiast mniej istotna jest szybkość wykonania, która często zależy od szybkości wprowadzania danych przez użytkownika. Niebagatelną rolę odgrywa też znacznie niższa cena systemu MRI. Z drugiej strony, dla użytkowników wytwarzających oprogramowanie na sprzedaż prawdopodobnie lepszy będzie system Logitech, zwłaszcza że od wersji 1.10 zawiera program usługowy LOD2EXE, dokonujący konwersji plików typu LOD na pliki typu EXE systemu PC-DOS. Omówione właściwości czynią go odpowiednim zarówno do poważnej produkcji oprogramowania, jak i do nauczania początkowego.

Oprac. JZ

Atrybuty jakości oprogramowania...

dokończenie z III s. okładki

Strukturalność (ang. *structuredness*) – właściwość kodu polegająca na jego podziale na niezależne części według ściśle określonego wzorca. Wymaga to projektowania programu w sposób uporządkowany i systematyczny, np. metodą zstępującą lub dekompozycji funkcji, oraz kodowania za pomocą dobrze określonych struktur sterujących, np. pętli *DO-WHILE* i instrukcji warunkowych *IF-THEN-ELSE*.

Śladowalność (ang. *traceability*) – atrybut oprogramowania umożliwiający śledzenie jego rozwoju, od specyfikacji wymagań aż do implementacji, w określonym środowisku produkcyjnym i eksploatacyjnym.

Testowalność (ang. testability) – stopień w jakim kod ułatwia tworzenie planu testowania, specyfikacji testów, procedur testujących i ich realizowanie. W praktyce sprzyja temu modularna budowa programu i dobrze określone sprzężenia, które mogą być testowane niezależnie.

Użyteczność (ang. *usability*) – wysiłek wymagany do nauczenia się, eksploatowania, przygotowywania danych i interpretowania wyników programu.

Wielokrotnego użycia (ang. *reusability*) – stopień w jakim program lub jego części mogą być używane w innych zastosowaniach. Właściwość ta wiąże się z zakresem funkcji wykonywanych przez program i ich obudowaniem.

Współdziałanie (ang. *interoperability*) – wysiłek wymagany do sprzęgnięcia systemu z innym niezależnym systemem.

Zdolność obsługi błędów (ang. *error handling capability*) – dostosowanie kodu do reagowania na awarie sprzętu lub oprogramowania i błędne polecenia operatora w taki sposób, aby wydajność systemu nie spadała gwałtownie, lecz łagodnie.

Zrozumiałość (ang. understandability) – stopień w jakim funkcje kodu są jasne dla czytelnika. W praktyce uzyskuje się to przez niesprzeczne użycie nazw i symboli, samoopisywalność modułów, uproszczenie struktur sterujących i dostosowanie ich do określonych norm itp. Kod nie powinien zawierać ukrytych znaczeń i charakterystyk, które stają się jasne dopiero po dłuższym użyciu.

Zupełność (ang. *completeness*) – właściwość kodu polegająca na tym, że wykonuje on wszystkie wymagane funkcje w sposób pełny. Wynika stąd, że dostępna jest pełna dokumentacja, a wymagane funkcje są zakodowane zgodnie z projektem.

Zwięzłość (ang. *consiseness*) – właściowość kodu polegająca na zrealizowaniu wymaganych funkcji w sposób maksymalnie oszczędny pod względem długości kodu, przy braku elementów zbędnych i nadmiarowych.

JANUSZ ZALEWSKI

Sieci komputerowe najbardziej dynamiczną gałęzią gospodarki

Liczący około czterdziestu lat przemysł komputerowy stał się po motoryzacyjnym i naftowym trzecim przemysłem USA, z rocznym obrotem około 150 mld dolarów. Zachodzą w nim zmiany jakościowe, dostosowujące go do wymagań użytkowników. Najbardziej charakterystyczne zmiany to znaczne zwiększenie możliwości mikrokomputerów, które mogą realizować wiele spośród funkcji minikomputerów, a nawet dużych komputerów, oraz powszechne żądanie określania przez wytwórców metod współpracy systemów ze sobą. Dlatego obserwuje się wysiłek producentów komputerów, nie tylko amerykańskich, zmierzający do opracowania systemów pozwalających komputerom różnej wielkości łatwo komunikować się przez rozległe często sieci. Firmy nawet współpracują ze sobą w tej dziedzinie, co było nie do pomyślenia kilka lat temu.

Korzystają z tego z jednej strony silne firmy mikrokomputerowe, jak Apple i IBM, a z drugiej – firmy specjalizujące się w łączeniu różnych systemów, jak Digital Equipment Corporation (DEC) i Hewlett-Packard. Korzyści odnoszą też użytkownicy, osiągając lepsze rezultaty z poczynionych inwestycji w sprzęcie i oprogramowaniu. Tempo usprawniania mikrokomputerów nie słabnie, dzięki czemu mogą być one stosowane do coraz bardziej złożonych zadań. Jeszcze szybszy jest postęp w rozwoju sieci. Przykładem tego może być sieć komputerowa przedsiębiorstwa Apollo Computer w Chelmsford (stan Massachusetts), gdzie ponad trzy tysiące pracowników może korzystać z komputerów dziewięciu różnych wytwórców, a także z ich programów i danych, uzyskując natychmiast raporty z odległych oddziałów.

Coraz częściej pojawiają się złożone mikrokomputery wyposażone w podzespoły sieciowe. Ocenia się, że w ciągu czterech lat osiągnęły one 10% produkcji przemysłu komputerowego i rozwijają się szybciej niż inne dziedziny tego przemysłu. Na początku lat dziewięćdziesiątych systemy te będą miały decydujący wpływ na rozwój tego przemysłu. W wyniku tego zwiększył się popyt na skomputeryzowane stanowiska pracy oferowane, na przykład, przez firmy Apollo i Sun Microsystems.

Ponadto rozwój sieci obalił zasadę, że klient powinien być związany z jednym dostawcą, ze względu na nakłady ponoszone na sprzęt, oprogramowanie i szkolenie. Z chwilą gdy stało się możliwe łatwe łączenie sprzętu różnego rodzaju, użytkownicy mogą dobierać sobie najkorzystniejsze dla nich elementy systemu u różnych wytwórców. Do sieci opłaca się kupować drogie, wyspecjalizowane superkomputery; dowodem może być firma General Motors, gdzie 300 inżynierów korzysta z komputera Cray do badań aerodynamicznych i symulowania wypadków. Wraz z upowszechnieniem łączy danych, sieci obejmują nowe dziedziny, jak automatyzacja biur, gdzie wyspecjalizowane komputery nadzorują wykorzystanie drukarek, kierują pocztą elektroniczną itp.

Żaden z wytwórców nie jest w pełni przygotowany do tej sytuacji. Stąd wspólne przedsięwzięcia, np. firmy DEC, która od dawna zajmuje się sieciami, z takimi firmami jak Apple i Compaq. Nawet potężna IBM, z własnymi rozwiązaniami niemal w każdej dziedzinie, bierze udział w konsorcjum mającym na celu rozwiązanie problemów niezgodności systemów komputerowych.

Ocenia się, że przedsiębiorstwa amerykańskie zakupiły ponad 15 milionów komputerów osobistych. Do sierpnia 1988 r. mniej niż 30% spośród nich było połączonych w sieci, lecz według B. Lyonsa, dyrektora handlowego systemów osobistych IBM, do roku 1991 w sieciach będzie pracować 90–95% tych komputerów. Dlatego mikrokomputery PS/2 tej firmy są wyposażone w tzw. magistralę Micro Channel, ułatwiającą połączenie z minikomputerami i dużymi komputerami IBM. W przyszłości będą one mogły działać jako terminale dużych systemów przetwarzania danych. Dotychczas funkcję tę spełniały zwykłe terminale, które nie miały możliwości samodzielnego przetwarzania.

W USA jest obecnie ponad 20 milionów terminali. Firmy mikrokomputerowe z Apple na czele mają nadzieję przejąć poważną część rynku tych urządzeń. Przewidywania te są oparte na właściwościach najnowszych mikrokomputerów, które są łatwe w obsłudze i umożliwiają dołączenie do sieci. Firma Apple dominowała w 1981 roku na rynku mikrokomputerowym obejmując wówczas 26,1% sprzedaży. Przed czterema laty przeżyła kryzys spowodowany zbyt optymistyczną oceną sytuacji i nagromadzeniem nadmiernych zapasów. W ciągu sześciu lat (1981–1987) wartość sprzedanych w USA mikrokomputerów zwiększyła się dziewięciokrotnie, do 13,5 mld dolarów i na czołowym miejscu znalazł się IBM (36,3% sprzedaży), a na drugim Apple (16,9%). Ta ostatnia ma obecnie dobre perspektywy – przewidywania na rok 1988 mówią o 40% wzroście, do 3,8 mld dolarów.

W tym samym okresie dochody ze sprzedaży minikomputerów zwiększyły się tylko o 76%, do 19,4 mld dolarów. Tutaj z kolei zamieniły się na czołowych miejscach firmy IBM i DEC; ta ostatnia w 1987 roku miała 24,9% wartości sprzedaży na rynku minikomputerów, wobec 17,3% w roku 1981 (odpowiednie wskaźniki dla IBM wynoszą 16,4 i 24,5%). W dziedzinie dużych komputerów wzrost był jeszcze wolniejszy (67%, do 23,3 mld dolarów), przy czym cały czas dominuje tu IBM z nieznacznym tylko spadkiem udziału (z 74,7% w 1981 roku do 70,4% w 1987).

Według oceny Gartner Group systemy komunikacyjne będą w 1992 roku tworzyć rynek o wartości 18 mld dolarów i zapotrzebowanie na nie będzie się gwałtownie zwiększać. Dlatego duże firmy zakupują małe przedsiębiorstwa specjalizujące się w systemach sieciowych. Przykładem tego jest nabycie w 1987 r. przedsiębiorstwa Timeplex przez trzecią amerykańską firmę komputerową Unisys. Firma DEC, która zajmuje się sieciami od 15 lat ma również własną sieć łączącą 27 tysięcy komputerów w 66 krajach, w ramach której uzyskuje się dostęp do mocy obliczeniowej. Nie jest to jednak sieć tak uniwersalna, jak wspomniana uprzednio sieć Apollo – nie zawsze można w niej pracować ze sprzętem innych producentów.

Zastosowanie sieci komputerowych, oprócz zwiększenia efektywności i oszczędności na szkoleniu czy oprogramowaniu, pozwoli połączyć niezależnie działające ośrodki i kierować specjalizowanym przetwarzaniem informacji w dużych przedsiębiorstwach. Jednym z problemów, jakie napotyka się w rozwoju sieci komputerowych, jest zasada poufności oraz możliwość niszczenia użytecznych programów i danych przez inne programy. Sieci mogą pośredniczyć w przenoszeniu tych ostatnich na inne komputery, dlatego eksperci przewidują w przyszłości opracowanie programów wykrywających te zajwiska i chroniących przed ich powstawaniem.

Dane finansowe głównych firm komputerowych w USA

Firma	Wpływy 1987–1988 r. (mld dolarów)	Sprzedaż sprzętu (mld dolarów)			Cena jednej akcji		Procentowa zmiana
		mikrokomputery	minikomputery	duże komputery	1.07.1987 r.	1.07.1988 r.	
IBM	55,3	4,9	3,2	16,4	164	127	−22,6
DEC	10,8	0,3	4,8	0,4	165	114	−30,9
Unisys	9,7	0,2	1,1	2,7	41	36	−12,2
Hewlett-Packard	,9,0	0,6	1,9	–	60	55	− 8,3
Apple	3,3	2,3	–	–	40	47	+17,5

Ponadto realizacja sieci okazuje się często złożonym zadaniem, wciąż brak jest jednolitych standardów zarówno wśród sprzętu, jak i oprogramowania. Jeśli chodzi na przykład o systemy operacyjne, to chciano zastosować tu Unix, który działa na różnych komputerach. Jednak umowa z 1987 roku między AT&T a Sun Microsystem spowodowała, że firmy konkurencyjne, z DEC, Hewlett-Packard i IBM na czele, utworzyły *Open Software Foundation* w celu opracowania systemu alternatywnego. Obie strony reklamują swoje rozwiązania.

Od dawna podejmuje się wysiłki mające na celu opracowanie norm sieciowych. Zajmują się tym 23 międzynarodowe organizacje normalizacyjne, w tym specjalnie powołane Zrzeszenie Systemów Otwartych *(Corporation for Open Systems)*, złożone z 17 głównych producentów komputerowych i kilkudziesięciu użytkowników. W czerwcu 1988 roku demonstrowano pierwsze wersje tych norm na konferencji Enterprise'88 w Baltimore, lecz nie zadowoliły one ogółu użytkowników, którzy chcieliby już z nich korzystać. Eksperci oczekują, iż pełny zakres opartych na normach wyrobów będzie osiągnięty dopiero za kilka lat.

Niektóre z dużych firm próbują oferować własne rozwiązania. IBM opracowuje swoją SAA *(Systems Application Architecture* – architektura zastosowań systemowych, umożliwiającą dołączanie wyrobów innych wytwórców oraz integrację różnych własnych rodzin komputerowych. Przedsięwzięcie to powinno umożliwiać współpracę wszystkich systemów. Najnowsze wyroby tej firmy, jak komputery osobiste PS/2 i minikomputery AS/400, są dostosowane do pracy w sieciach i mogą korzystać z tego samego oprogramowania. Również DEC i Hewlett-Packard pracują nad dostosowaniem swoich sieci do wyrobów innych producentów, zwłaszcza IBM.

Użytkownicy mają nadzieję, że uniwersalna sieć standardowa umożliwi wymienne stosowanie konkurencyjnych komputerów. Najwięksi klienci, jak rząd USA, który kupuje głównie komputery za 17 mld dolarów, General Motors lub Boeing są za „otwartymi" normami. Zdarza się jednak, jak np. w przetargu ogłoszonym w 1987 r. przez Siły Powietrzne USA, że jest wymagany sprzęt pracujący pod kontrolą Unixa. Firma DEC wycofała się wówczas z negocjacji, ale mniejsze firmy dostosowały się do żądań.

W epoce sieci użytkownicy kupują przede wszystkim mikrokomputery. Konkurencja na rynku dużych i wyspecjalizowanych komputerów jest bardzo silna i tylko duże przedsiębiorstwa mają tu coś do powiedzenia. Stąd tendencja do łączenia się, czego przykładem jest Unisys powstały w 1986 roku z firm Borroughs i Sperry.

Na zakończenie zestawiono finansowe dane pięciu głównych amerykańskich firm informatycznych. Jak widać, istnieje tu specjalizacja: IBM i Unisys dominują w produkcji dużych komputerów, Dec i Hewlett-Packard w mini-, a Apple mikrokomputerów. Tylko tej ostatniej udało się, mimo krachu giełdowego z jesieni 1987 roku, zwiększyć cenę swych akcji. Można przewidywać, że m.in. dzięki rozwojowi sieci, w najbliższych latach Apple znacznie się rozwinie.

Opracował **JAN RYŻKO**
na podstawie Fortune

Bardzo prosimy Autorów przysyłających teksty artykułów do opublikowania w INFORMATYCE o podawanie danych umożliwiających bezpośredni kontakt w godzinach pracy (numer telefonu) oraz adres domowy (do korespondencji i wysłania honorarium).

COIG

CENTRALNY OŚRODEK INFORMATYKI GÓRNICTWA
40-065 KATOWICE UL. MIKOŁOWSKA 100
telefon – 574-777 telefaks – 517-442 teleks – 00313711

OFERUJE USŁUGI W ZAKRESIE:

- projektowania i oprogramowania systemów informatycznych;
- ujmowania i przetwarzania danych na komputerach MERA–9150, ODRA–1300, ICL 1900, 2900, 39, RIAD;
- kompletacji i sprzedaży systemów mikrokomputerowych, kopiarek i telekopiarek;
- instalacji, napraw i przeglądów okresowych urządzeń mikrokomputerowych, kopiarek i telekopiarek;
- szkolenia projektantów, programistów i operatorów sprzętu informatycznego.

OFERTY NASZE OBEJMUJĄ:

- udostępnienie eksploatowanych rozwiązań informatycznych oraz oprogramowania narzędziowego;
- projektowanie systemów informatyczych według założeń użytkowników lub na podstawie analizy jego potrzeb;
- ujmowanie danych źródłowych i ich przetwarzanie w trybie wsadowo-partiowym oraz zdalnym;
- udostępnianie czasu pracy maszyn cyfrowych;
- instalacje sieci lokalnych (Lan-Link, D-Link, Oa-Link);
- kompletacje i dostawę sprzętu mikrokomputerowego oraz kopiarek i telekopiarek;
- serwis techniczny gwarancyjny i pogwarancyjny sprzętu informatycznego realizowany w ciągu 48 godzin od wezwania klienta, dostosowanie drukarek D–180 do współpracy z IBM PC/XT/AT, przeróbki powiększające liczby znaków drukowanych do 158 w wierszu, rozszerzanie pamięci mikrokomputerów, wprowadzanie polskich liter do generatorów znaków;
- szkolenie informatyków w ośrodku wyposażonym w bogaty sprzęt dydaktyczny.

EO/943/89

European CAMAC Associaton

European CAMAC Association (ECA) jest międzynarodowym stowarzyszeniem prawnie zarejestrowanym w Belgii, działającym na polu przetwarzania danych czasu rzeczywistego i sterowania procesami. Obecnie, koncentruje działalność w trzech dziedzinach:

- opracowywanie norm i zaleceń do zastosowań w systemach czasu rzeczywistego,
- popieranie stosowania takich norm,
- nadzór nad przestrzeganiem norm, w których tworzeniu uczestniczyło stowarzyszenie.

Przetwarzanie danych czasu rzeczywistego i sterowanie w rozumieniu norm ECA to dziedzina interakcyjnego zbierania danych pochodzących z doświadczeń i procesów przemysłowych, a także sterowanie tymi doświadczeniami i procesami w czasie rzeczywistym.

ECA jest młodszą siostrą Komitetu ESONE (ang. *European Standards on Nuclear Electronics*), który stworzył pełny zestaw norm sprzęgania urządzeń komputerowych w systemie CAMAC. W normach tych opisano zasady tworzenia sprzężeń w kategoriach specyfikacji elektrycznej, logicznej i mechanicznej. ESONE zrzesza przedstawicieli europejskich laboratoriów, organizacji rządowych, wyższych uczelni i innych instytucji, które nie mają na celu osiągania zysków z prowadzonej działalności, lecz są zainteresowane współpracą w rozwiązywaniu problemów związanych ze stosowaniem norm i zaleceń praktycznych w systemach elektronicznych. W przeszłości główne zadanie ESONE polegało na określaniu norm systemu CAMAC. Obecnie większość działań jest skierowana na organizację wymiany doświadczeń między członkami i definiowanie wspólnych zaleceń. Prace prowadzone są w następujących grupach studialnych:

- systemy zaawansowane (Fastbus),
- komputerowe wspomaganie projektowania w elektronice (ECADE),
- magistrala VME,
- komputery osobiste,
- oprogramowanie czasu rzeczywistego,
- pielęgnowanie dokumentacji (wspólne przedsięwzięcie ESONE i ECA).

Działalność ECA uzupełnia pracę Komitetu ESONE. ECA zrzesza 66 członków z całej Europy: osoby prywatne, instytucje i stowarzyszenia krajowe, które istnieją w Danii, Wielkiej Brytanii i Polsce. Z reguły są to laboratoria badawcze i firmy dostarczające sprzęt i oprogramowanie z dziedziny przetwarzania danych czasu rzeczywistego. Stowarzyszeniem zarządza dwudziestopięcioosobowa Rada, której członkowie są wybierani z zachowaniem reguł gwarantujących odpowiednią reprezentację poszczególnych krajów w Zgromadzeniu Ogólnym, odbywającym się co pół roku. Rada wybiera Zarząd, w skład którego wchodzi przewodniczący, dwóch wiceprzewodniczących, sekretarz i skarbnik. Jego działalność polega na nadzorowaniu bieżącej działalności inicjowanej przez Radę. Obecnie, w skład Rady wchodzą przedstawiciele Austrii, Belgii, Danii, Francji, Holandii, Norwegii, Polski, RFN, Szwajcarii, Węgier i Wielkiej Brytanii.

Przed rokiem 1980 działalność ECA koncentrowała się na promocji systemu CAMAC poza obszarem laboratoriów naukowych, gdzie CAMAC został stworzony. Wyrażało się to organizowaniem konferencji i akcji publikacyjnych. W tym czasie odbyły się trzy konferencje, zorganizowane we współpracy z innymi zainteresowanymi organizacjami, głównie ESONE:

- Pierwsze międzynarodowe sympozjum CAMAC, Luksemburg, 1973 r.
- Drugie międzynarodowe sympozjum CAMAC, Bruksela, 1975 r.
- Sympozjum przetwarzania danych i sterowania procesami w czasie rzeczywistym, Berlin Zachodni, 1979 r.

Do tej pory jest wydawany przewodnik *CAMAC Product Guide*, zawierający informacje o wszystkich urządzeniach systemu CAMAC dostępnych w handlu. Informacje o modułach zawarte w przewodniku są pogrupowane według specyfikacji funkcjonalnej. Opracowano również bibliografię artykułów na temat użycia systemu CAMAC.

Od około 1980 r., kiedy pozycja systemu CAMAC została ugruntowana, działalność stowarzyszenia skupiono na określeniu dziedzin przetwarzania danych w czasie rzeczywistym, szczególnie wymagających opracowania norm i zaleceń. Zorganizowano kilka dalszych konferencji, wspólnie z ESONE i Europejskim Warsztatem Przemysłowych Systemów Komputerowych – EWICS;

- Sympozjum przetwarzania danych i sterowania procesami w czasie rzeczywistym, Versailles, 1982 r.,
- Europejskie seminarium sieci lokalnych, Berlin Zachodni, 1983 r.,
- Europejskie seminarium sieci lokalnych, Bruksela, 1983 r.

Dwa ostatnie spotkania okazały się być bardzo pożytecznym forum dyskusyjnym sieci lokalnych, ze względu na wprowadzenie ich w wielu instytucjach badawczych. W 1983 r. rozpoczęto prace nad przewodnikiem powszechnie używanych sprzęgów sprzętowych – *Guide of Internationally Recognised Hardware Interfaces*. Celem tych prac było opublikowanie przewodnika zawierającego wprowadzenie do wszystkich istniejących norm nt. systemów sprzęgających, na tyle pełne, aby czytelnik mógł dokonać wyboru spośród istniejących urządzeń zgodnie ze swoimi potrzebami.

Po dwuletnim okresie uśpienia (od 1983 do 1985 r.) ECA wznowiło działalność realizując kontrakt z Komisją Wspólnot Europejskich CEC. Zorganizowano dwa seminaria:

- Oprogramowanie przenośne dla mikroprocesorów, Bruksela, 1986 r.,
- Normy i sprzęgi dla komputerów osobistych, Bruksela, 1987 r.

Pierwsze seminarium było ważnym krokiem na drodze do stworzenia grupy studialnej ESONE. Grupa ta obecnie zbiera informacje o istniejących programach i planuje wydanie ich przeglądu. Następnym krokiem będzie zdefiniowanie zestawu minimalnych wymagań wobec przyszłych systemów programowania. Jednym z ostatnich wydarzeń było wspólne spotkanie w 1987 r., w Paryżu obu organizacji ECA i ESONE, podczas którego odbyło się seminarium nt. CAD, jak również techniczna prezentacja działalności laboratoriów członkowskich.

Opracował M. KUC
na podstawie IES News

Nowe zasady przyjmowania ogłoszeń

Uprzejmie informujemy, że zlecenia na umieszczanie ogłoszeń w INFORMATYCE przyjmuje w bieżącym roku bezpośrednio nasza redakcja. W ten sposób nastąpiła eliminacja dotychczasowego pośrednictwa Biura Ogłoszeń. Obecnie każdy zainteresowany umieszczeniem ogłoszenia w INFORMATYCE uzyskuje nieporównywalnie lepsze możliwości natychmiastowego wyjaśnienia oraz uzgadniania wszystkich spraw natury technicznej, organizacyjnej i finansowej.

Zapytania o warunki i aktualne ceny, a także konkretne zamówienia prosimy więc kierować bezpośrednio do naszej redakcji, która na żądanie wyśle upoważnioną osobę w celu dokonania na miejscu wiążących ustaleń.

Przy okazji pragniemy zwrócić uwagę wszystkich zainteresowanych umieszczaniem ogłoszeń w naszym miesięczniku, że jesteśmy od 25 lat w dalszym ciągu jedynym profesjonalnym periodykiem krajowym naukowo-technicznym, docierającym do całego środowiska informatycznego. Oznacza to, że z punktu widzenia zainteresowań potencjalnego zleceniodawcy ogłoszeń sprzętu komputerowego, oprogramowania, materiałów pomocniczych czy usług, ogłoszenie w INFORMATYCE będzie bardziej skuteczne od jego umieszczenia w innych wydawnictwach.

Oczekujemy na zgłoszenia. Nasz adres: 01–552 Warszawa, Pl. Inwalidów 10, p. 128, 136, telefon 39-14-34.

Redakcja

Język lotos (2)

dokończenie ze str. 24

Dodatkowo przyjęto założenia upraszczające, że interakcje są wyłącznie typu synchronizującego, czyli nie przenoszą żadnej informacji między procesami, oraz że w definicji procesów nie występują parametry związane z przekazywaniem wartości.

Podany zestaw aksjomatów i reguł wyznacza pewien system, który można wykorzystywać do generowania dowodów, czyli w dotychczas używanej terminologii, do generowania obliczeń. Łatwo przekonać się, że prezentowane poprzednio przykłady obliczeń, po uwzględnieniu przyjętych założeń upraszczających, są zgodne z tym systemem.

Ograniczając się do pierwszych trzech przykładów łatwo stwierdzić, że zbiory ciągów interakcji związane z obliczeniami omawianych procesów przedstawiają się następująco:

- dla procesu *Bufor*[*a*, *b*] z przykładu 1 jest to zbiór ciągów skończonych postaci:

$\{\langle ax, bx\rangle | x \in integer\}$;

- dla procesu *Buf*[*a*, *b*] z przykładu 2 zbiór ciągów nieskończonych:

$\{\langle a\,x1, b\,x1, a\,x2, b\,x2\,...\rangle |\, x1, x2, ... \in integer\}$;

- dla procesu *Buf*1[*a*, *b*] (10) z przykładu 3 zbiór

$\{\langle bx\ 10, a\,x1, b\,x1, a\,x2, b\,x2, ...\rangle |\, x1, x2, ... \in integer\}$.

LITERATURA

[1] Brinksma E. (Ed.): Information processing systems – Open systems interconnection – Lotos – A formal description technique based on the temporal ordering of observational behaviour. ISO 8807, ISO/TC97/SC16, Geneva, 1988

[2] Pnueli A.: Specification and development of reactive systems. Information Processing 86, H. J. Kugler (ed.), pp. 845 858, North-Holland, Amsterdam, 1986.

Nowe książki

Wydawnictwo Czasopism i Książek Technicznych NOT SIGMA

Andrzej Markowski, Andrzej Rydzewski, Henryk Kruszyński: Leksykon techniki mikrokomputerowej. Warszawa 1989, wydanie I, nakład 30 tys. egz., s. 120, cena 890 zł, ISBN 83–85001–28–X

Leksykon jest przeznaczony dla szerokiego kręgu Czytelników nie mających przygotowania z zakresu informatyki. Zawiera ponad 500 haseł z definicjami w języku polskim, przy czym przy terminach polskich podano odpowiedniki angielskie, niemieckie, francuskie oraz rosyjskie. Autorzy starali się, aby leksykon był zwarty, stąd ograniczona liczba haseł i unikanie rozbudowanych haseł wieloczłonowych. Na końcu książki zamieszczono słowniki alfabetyczne w językach obcych oraz Dodatek z zestawieniem kodów.

Henryk Kruszyński, Krzysztof Kulpa: Mikroprocesor 6502 i jego rodzina. Warszawa 1989, wydanie I, nakład 19 350 egz., s. 160, cena 1100 zł, ISBN 83–85001–58–1

Książka prezentująca rodzinę procesorów 65XX składa się z dwu części. W pierwszej omówiony jest mikroprocesor 6502, jego budowa, zasada działania i sposoby programowania. Część druga, wymagająca od Czytelnika pewnego zasobu wiedzy z dziedziny elektroniki, a szczególnie układów logicznych (cyfrowych), omawia budowę mikrokomputera, którego sercem jest procesor z rodziny 65XX, a także działanie poszczególnych jego bloków i sposób programowania. Książka może być przydatna osobom, które samodzielnie pragną skonstruować mikrokomputer lub dokładnie zrozumieć szczegóły budowy gotowych komputerów.

DOŚWIADCZONA, DYNAMICZNA SPÓŁKA MIĘDZYNARODOWA z udziałem kapitału amerykańskiego i zachodnioniemieckiego

ZATRUDNI

informatyków i programistów w Europie Zachodniej.

Wymagana biegła znajomość języka zachodniego i profesjonalizm. Oczekuje się zdolności wykonywania pracy obejmującej uzgodnienia z klientami, projekt i aplikację systemu.

OFERTY ze szczegółowym opisem praktyki zawodowej w języku angielskim lub niemieckim oraz kserokopię dyplomu prosimy składać pod adres:

OFERTA–I
skr. poczt. 105,
00–971 Warszawa

ZAPEWNIAMY DYSKRECJĘ.

EO/1139/89

Kubicki W., Suskiewicz M.: Możliwości stosowania światłowodów w krajowych systemach informatycznych
INFORMATYKA 1990, nr 2, s. 1
Charakterystyka aktualnych możliwości stosowania światłowodów w abonenckich oraz lokalnych sieciach komputerowych w Polsce z użyciem konwerterów światłowodowych.

Kubicki W., Suskiewicz M.: Fiber-optics application possibilities in national data processing systems
INFORMATYKA 1990, No. 2, p. 1
Characteristics of actual possibilities for fiber-optics application in subscriber and local area computer networks in Poland using fiber-optics converters.

Kubicki W., Suskiewicz M.: Anwendungsmöglichkeiten der Lichtwellenleiter in inländischen EDV-Systemen
INFORMATYKA 1990, Nr. 2, S. 1
Eine Charakteristik der heutigen Anwendungsmöglichkeiten von Lichtwellenleiter in Teilnehmer- und Lokalcomputernetzen in Polen mit Einsatz von Lichtwellenleiterumwandlern.

Runka H. J.: Długi wektor w Turbo Pascalu 5.0
INFORMATYKA 1990, nr 2, s. 3
Praktyczne sposoby wykorzystywania długiego wektora liczb w języku Turbo Pascal 5.0.

Runka H. J.: The long vector in Turbo Pascal 5.0
INFORMATYKA 1990, No. 2, p. 3
Practical methods for using the long number vector in the Turbo Pascal 5.0 language.

Runka H. J.: Der lange Vektor in Turbo Pascal 5.0
INFORMATYKA 1990, Nr. 2, S. 3
Praktische Methoden für Anwendung des langen Zahlenvektors in der Turbo Pascal 5.0-Sprache.

Ormiński M.: Ethernet – od specyfikacji do realizacji praktycznej (1). Specyfikacja
INFORMATYKA 1990, nr 2, s. 6
Pierwsza część charakterystyki rozwiązań oraz sposobu działania sieci Ethernet.

Ormiński M.: The Ethernet – from specification to practical realization (1). The specification
INFORMATYKA 1990, No. 2, p. 6
First part of characteristics of the Ethernet network solutions and operating method.

Ormiński M.: Ethernet – von Spezifikation bis praktischer Realisation (1). Die Spezifikation
INFORMATYKA 1990, Nr. 2, S. 6
Erster Teil einer Charakteristik von Lösungen und Betriebsweise des Ethernest-Netzwerkes.

Kłopotek M. i inni: SYS8688 – system pozyskiwania i weryfikacji wiedzy medycznej (1)
INFORMATYKA 1990, nr 2, s. 9
Pierwsza część charakterystyki rozwiązań programowych oraz sposobu działania systemu ekspertowego dla wybranych obszarów medycyny, zawierająca omówienie rozwiązań systemów bazy danych oraz bazy wiedzy.

Kłopotek M. et al.: SYS8688 – a system for medical science acquisition and verification (1)
INFORMATYKA 1990, No. 2, p. 9
First part of characteristics of program solutions and operation method for an expert system in selected medicine areas, which includes discussion of data base and science base system solutions.

Kłopotek u.a.: SYS8688 – ein System für Gewinnung und Verifikation der Medizinwissenschaft (1)
INFORMATYKA 1990, Nr. 2, S. 9
Erster Teil einer Charakteristik von Programmlösungen und Betriebsweise eines Expertensystea für ausgewählte Medizinbereiche, der eine Besprechung von Lösungen der Daten- und Wissenschaftsbanksysteme umfasst.

Banachowski L., Deminet J., Lao M.J.: Propozycja nowego programu uniwersyteckich studiów informatycznych (1)
INFORMATYKA 1990, nr 2, s. 13
Pierwsza część szczegółowego omówienia propozycji nowego programu uniwersyteckich studiów informatycznych w Polsce, wzorowanego na rozwiązaniach zalecanych przez wiodące światowe organizacje informatyczne.

Banachowski L., Deminet J., Lao M.J.: A proposal for a new program of informatics university studies (1)
INFORMATYKA 1990, No. 2, p. 13
First part of detailed discussion of the new program proposal for informatics university studies in Poland, which is modelled after solution recommended by leading informatics worldwide organizations.

Banachowski L., Deminet J., Lao M.J.: Ein Vorschlag des neuen Programmes für Informatik-Universitätsstudien (1)
INFORMATYKA 1990, Nr. 2 S. 13
Erster Teil einer detaillierten Besprechung des Vorschlages von neuem Programm für Informatik-Universitässtudien in Polen, das sich nach Empfehlungen der führenden Informatikweltorganisationen bildet.

Zalewski J.: Okres istnienia oprogramowania (2)
INFORMATYKA 1990, nr 2, s. 18
Druga część omówienia norm amerykańskich dotyczących organizacji procesu wytwarzania produktów programowych.

Zalewski J.: Software life cycle (2)
INFORMATYKA 1990, No. 2, p. 18
Second part of discussion of american standards for organization of program products manufacturing process.

Zalewski J.: Lebenszyklus der Software (2)
INFORMATYKA 1990, nr 2, s. 18
Zweiter Teil einer Besprechung von amerikanischen Normen aus dem Bereiche der Organisation des Programmprodukteerzeugungsprozesses.

Huzar Z., Kuźniarz L.: Język Lotos (2). Specyfikacja procesów
INFORMATYKA 1990, nr 2, s. 21
Druga część charakterystyki języka Lotos jako narzędzia specyfikacji formalnej, zawierająca omówienie sposobu specyfikowania procesów.

Huzar Z., Kuźniarz L.: The Lotos language (2). Process specification
INFORMATYKA 1990, No. 2, p. 21
Second part of the Lotos language characteristics as a tool for formal specification, which includes discussion of the metod for process specification.

Huzar Z., Kuźniarz L.: Lotos-Sprache (2). Die Prozessespezifikation
INFORMATYKA 1990, Nr. 2, S. 21
Zweiter Teil einer Charakteristik der Lotos-Sprache als ein Werkzeug für Formalspezifikation, der eine Besprechung der Methode für Prozessespezifikation umfasst.

Atrybuty jakości oprogramowania – definicje TAEA

Nawiązując do pierwszej części artykułu nt. okresu istnienia oprogramowania (*Informatyka*, nr 1, 1990) przedstawiam w swobodnym tłumaczeniu określenia atrybutów jakości oprogramowania opisane w raporcie *"Manual on Quality Assurance for Computer Software"*, International Atomic Energy Agency, Vienna, April 1987.

Czytelność (ang. *readability*) – stopień w jakim na podstawie odczytania kodu można wnioskować o jego funkcjach i projekcie, co wyraża się – na przykład – przez użycie znaczących nazw zmiennych, zaopatrzenie kodu w obszerne komentarze itp.

Dokładność (ang. *accuracy*) – stopień w jakim wyniki wykonania kodu odpowiadają pierwotnym zamierzeniom.

Efektywność (ang. *efficiency*) – stopień w jakim kod, wykonując wymagane funkcje, wykorzystuje zasoby. Może to oznaczać konieczność wyboru odpowiednich konstrukcji języka źródłowego w celu uzyskania minimalnej objętości kodu wynikowego, zastosowanie takich algorytmów, które są najszybciej wykonywane, lub przyjęcie specjalnych metod upakowania informacji w pamięci.

Ergonomiczność (ang. *human engineering*) – stopień w jakim kod spełnia swoje funkcje nie powodując strat czasu i energii użytkowników oraz nie zmniejszając ich dobrego samopoczucia. W tym celu, dane wejściowe i wyniki powinny być zrozumiałe i jednoznaczne, powinny uniemożliwiać mylną interpretację.

Integralność (ang. *integrity*) – stopień w jakim poddany jest kontroli dostęp do oprogramowania lub danych przez nieupoważnione osoby.

Komunikatywność (ang. *communicativeness*) – stopień w jakim postać i treść danych wejściowych i wyników działania kodu mogą być zrozumiane i przyswojone. Atrybutem tym obejmuje się często także zrozumiałość i przyswajalność standardowych protokołów i sprzężeń służących do połączenia programu z innym niezależnym systemem.

Konserwowalność (ang. *maintainability*) – stopień w jakim kod jest podatny na aktualizację w celu spełnienia nowych wymagań, usunięcia wad lub przeniesienia do innego, lecz podobnego systemu komputerowego.

Modyfikowalność (ang. *modifiability*) – właściwość projektu lub kodu ułatwiająca wprowadzenie określonych zmian do tego projektu lub kodu.

Niesprzeczność (ang. *consistency*) – stopień, w jakim program można uważać za jednolity pod względem przyjętej notacji, terminologii, symboliki, komentarzy i technik realizacji.

Niezależność od urządzeń (ang. *device independence*) – zdolność kodu do uniknięcia skutków wymiany sprzętu komputerowego lub urządzeń zewnętrznych. W tym celu należy zidentyfikować, wyodrębnić i zminimalizować segmenty kodu odnoszące się bezpośrednio do określonych urządzeń.

Niezawodność (ang. *reliability*) – stopień w jakim kod jest zdolny do nieustannego wykonywania zamierzonych funkcji.

Odporność (ang. *robustness*) – stopień w jakim kod może kontynuować zamierzone działanie, mimo pogwałcenia niektórych założeń specyfikacji. Znaczy to, że program będzie przyjmował dane wejściowe lub wyniki pośrednie, które przekraczają zakresy, mają inny format lub typ niezgodny z założonym, bez ograniczenia swoich funkcji.

Poprawność (ang. *correctness*) – stopień w jakim program spełnia swoją specyfikację, co rozumie się także jako zdolność programu do uzyskiwania założonych wyników, przy założonych danych wejściowych.

Prostota (ang. *simplicity*) – właściwość oprogramowania polegająca na zaimplementowaniu jego funkcji w możliwie najbardziej zrozumiały sposób. Zwykle wymaga to unikania technik zwiększających złożoność.

Przenośność (ang. *portability*) – stopień w jakim kod może działać dobrze w konfiguracji komputerowej innej niż ta, dla której jest przeznaczony.

Rozliczalność (ang. *accountability*) – stopień w jakim można mierzyć użycie zasobów komputerowych przez moduł lub program. Znaczy to, że krytyczne segmenty kodu mogą być wyposażone w sondy, które mierzą czas wykonania, wykonywanie poszczególnych rozgałęzień itd. Segmenty wyposażone w sondy powinny być wcielane do kodu podczas kompilacji lub asemblacji warunkowej.

Rozszerzalność (ang. *augmentability*) – stopień w jakim można rozbudowywać kod pod względem właściwości funkcjonalnych lub zajętości pamięci.

Samoopisywalność (ang. *self-descriptiveness*) – stopień w jakim wydruk kodu zawiera informacje, na podstawie których czytelnik może określić lub zweryfikować jego cele, założenia, ograniczenia, dane wejściowe, wyniki, składowe i wersję.

Samowystarczalność (ang. *self-containedness*) – stopień w jakim kod wykonuje samodzielnie wszystkie swoje jawne i niejawne funkcje, np. inicjowanie, kontrole, diagnostykę itp.

dokończenie na s.27

Przedsiębiorstwo
Zastosowań Informatyki

meditronik

oferuje:

- systemy mikrokomputerowe
- programy aplikacyjne dla różnych dziedzin gospodarki (na życzenie wysyłamy katalog)
- poszukiwane komponenty elektroniczne
- interfejs do kamery video (opc. CCD) z bogatą biblioteką oprogramowania
- emulator Z80
- tester układów scalonych i pamięci
- programator BPROM
- asynchroniczny procesor komunikacyjny
- konwerter RS-232 – Centronics

instaluje:

- połączenia międzykomputerowe (XT/AT – ODRA/RIAD/IBM)
- systemy sieciowe (NOVELL)
- systemy wielodostępne (SCO Xenix 286, 386, Unix System V)

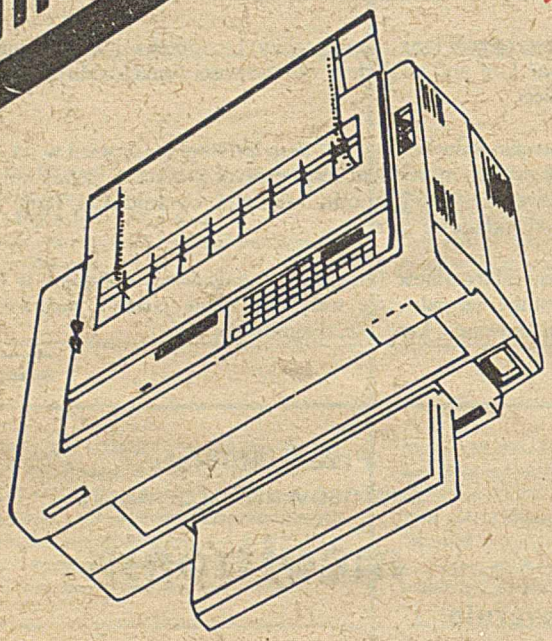
Jeżeli jesteś autorem oryginalnego programu aplikacyjnego
- skontaktuj się z nami, będziemy pośredniczyć w sprzedaży Twojego programu, dbając o ochronę Twoich praw autorskich!

Nasz adres:

00-194 Warszawa, ul. Dzika 4
tel. (02) 635-22-63, 635-22-64,
fax (02) 635-21-95
tlx 816075 medi pl

EO/605/89

SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA



Japoński koncern SEIKOSHA, produkujący znane zegarki SEIKO, migawki do fotoaparatów, drukarki komputerowe, zbliżył się do naszych rynków, otwierając filię w Hamburgu, produkującą pełną gamę drukarek komputerowych.

Informacje techniczne, katalogi:
SOFT-TRONIK SERVICE
00-710 Warszawa
ul. Idzikowskiego 2
tel. 40-46-79, fax (02) 635-21-95
telex 816075

EO/717/89